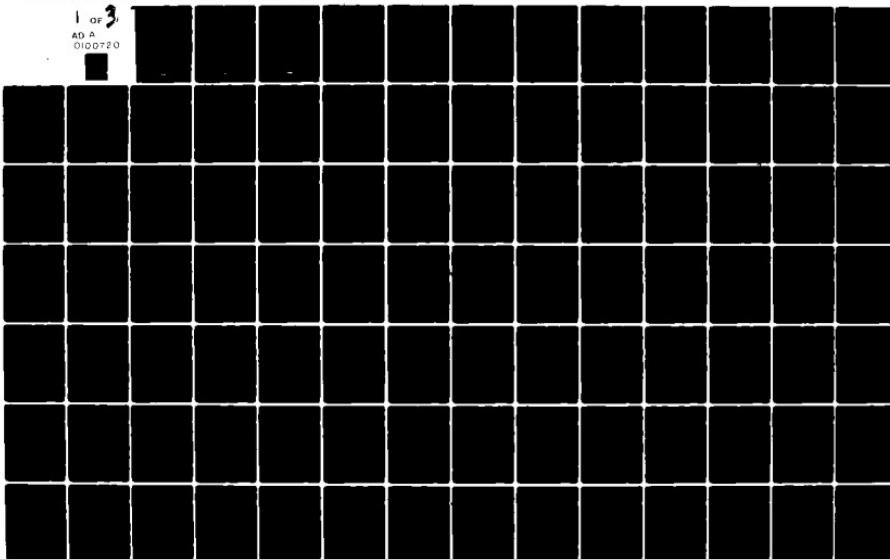


AD-A100 720

TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA
APPLICABILITY OF SRM TO THE VERIFICATION OF MANAGEMENT INFORMATION--ETC(U)
APR 81 R P LOSHBOUGH, M W ALFORD, J T LAWSON DAHC26-80-C-0020
UNCLASSIFIED TRW-37554-6950-001-VOL-1 NL

1 OF 3
AD A
0100720



37554-6950-001

12

ADA1000720

**APPLICABILITY OF SREM TO THE VERIFICATION
OF MANAGEMENT INFORMATION SYSTEM
SOFTWARE REQUIREMENTS**

**FINAL REPORT
Volume I**

CDRL A002

30 APRIL 1981

DTC FILE COPY

Prepared For

**U.S. Army Institute For Research and
Management Information Computer Science**

DAHC26-80-C-0020

JUN 30 1981

A

TRW
DEFENSE AND SPACE SYSTEMS GROUP
Huntsville, Alabama

This document has been approved
for public release and sale; its
distribution is unlimited.

81507029

14 TRW-37554-6950-001-V -

**APPLICABILITY OF SREM TO THE VERIFICATION
OF MANAGEMENT INFORMATION SYSTEM
SOFTWARE REQUIREMENTS. Volume I.**

9 FINAL REPORT,
Volume I

12 1/10

CDRL A002

11 30 APRIL 1981

10 R.P./Loshbough M.W./Alford J.T./L
D.M./Sims T.R./Jo.

Prepared For

U.S. Army Institute For Research and
Management Information Computer Science

15

DAHC26-80-C-0020

TRW
DEFENSE AND SPACE SYSTEMS GROUP
Huntsville, Alabama

4 1

APPLICABILITY OF SREM TO THE VERIFICATION
OF MANAGEMENT INFORMATION SYSTEM
SOFTWARE REQUIREMENTS

FINAL REPORT

VOLUME I

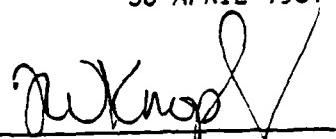
CDRL A002

30 APRIL 1981

Principal Authors:

R. P. Loshbough

Approved By:


F. W. Knopf, Manager
Software Engineering

Contributing Authors:

M. W. Alford

J. T. Lawson

D. M. Sims

T. R. Johnson

T. W. Thomas

Editor:

B. B. Bird


R. W. Godman, Manager
Huntsville Laboratory

Prepared For

Army Institute for Research and
Management Information Computer Science

*After m. file
for Doc's - ETT*

Under Contract
DAHC26-80-C-0020

A

TRW
DEFENSE AND SPACE SYSTEMS GROUP

TABLE OF CONTENTS

<u>Section</u>	<u>Title</u>	<u>Page</u>
1.0	INTRODUCTION.	1-1
1.1	SCOPE AND APPROACH OF THE SRM APPLICATION.	1-2
1.2	SUMMARY OF RESULTS.	1-4
1.3	OVERVIEW OF THIS REPORT	1-6
2.0	THE TRW SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY . .	2-1
2.1	PURPOSE AND BACKGROUND OF SRM.	2-2
2.2	SRM OVERVIEW	2-4
2.3	THE NEW PERSPECTIVE OF SRM	2-5
2.4	RANGE OF SRM ACTIVITIES.	2-9
2.5	SRM DEVELOPMENT CONSIDERATIONS	2-10
2.6	PRINCIPAL COMPONENTS OF SRM.	2-12
2.6.1	The Requirements Statement Language (RSL) .	2-12
2.6.2	The Requirements Engineering and Validation System (REVS)	2-15
2.6.3	The Application Methodology	2-21
2.7	SPECIFICATION MANAGEMENT.	2-27
2.7.1	Management Planning and Control	2-27
2.7.2	Review and Analysis	2-29
2.7.3	Partial List of Prespecified RADX Tests . .	2-30
2.7.4	Requirements Data Base Documentation. . .	2-32
2.7.5	Assessing the Impact of Requirements Changes	2-35
2.7.6	Software Test Planning.	2-36
2.8	MAINFRAME REVS SOFTWARE CONFIGURATIONS.	2-39
2.8.1	REVS Source	2-39
2.8.2	TRW PASCAL Development System	2-39
2.8.3	Data Base Control System (DBCS)	2-40
2.8.4	Compiler Writing System (CWS)	2-40
2.8.5	Ancillary Files	2-40
2.8.6	BMDATC ARC Installation	2-40
2.8.7	TRW DSSG Installation	2-41
2.8.8	NADC Installation	2-41
2.9	REVS IMPLEMENTATION ON THE DEC VAX 11/780	2-42

TABLE OF CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.0	DESCRIPTION OF THE SREM APPLICATION TO THE MOM DFSR . . .	3-1
3.1	INTRODUCTORY REMARKS.	3-1
3.2	SCOPE OF THIS EFFORT.	3-2
3.2.1	Approach to Overcoming Delays Caused by the High Error Count	3-2
3.2.2	Approach to Overcoming the High Input MESSAGE Count.	3-3
3.2.3	Impact of the Selected Approaches.	3-6
3.3	DEVELOPMENT OF INTERFACE ELEMENTS.	3-7
3.3.1	Subsystem Definition	3-8
3.3.2	Interface Definition	3-9
3.3.3	MESSAGE Definition	3-9
3.4	DEVELOPMENT OF GLOBAL DATA	3-23
3.5	DEVELOPMENT OF REQUIREMENTS NETS (R_NET)	3-31
3.5.1	R_NET Considerations.	3-31
3.5.2	Example of an R_NET Development	3-35
3.5.3	Definition of SUBNET Processing	3-37
3.5.4	Reason for Liberal Use of SUBNETs	3-40
3.5.5	Development of the ALPHA Description Sheet	3-41
3.5.6	Entry of R_NETs and SUBNETs Into the Requirements Data Base.	3-43
3.5.7	R_NETs as Source of Trouble Reports	3-44
3.6	DEVELOPMENT OF TRACEABILITY.	3-45
3.7	EVALUATION USING RADX.	3-50
3.7.1	The Premise of RADX Use	3-50
3.7.2	RADX Approach	3-51
3.8	ANALYSIS OF THE SREM APPLICATION EFFORT.	3-68
4.0	RESULTS OF SREM APPLICATION.	4-1
4.1	DESCRIPTION OF TROUBLE REPORTS	4-2
4.1.1	Trouble Report Format	4-2
4.1.2	Entry of Trouble Reports into the Requirements Data Base.	4-4
4.1.3	RADX Support of Management Review of Trouble Report.	4-6

TABLE OF CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
4.2	EVALUATION OF DISCREPANCIES.	4-8
4.2.1	DLT Considerations	4-8
4.2.2	R NET Contribution to Identification of DLT Deficiencies	4-10
4.2.3	Special SREM Procedures to Assure Unambiguous DATA Naming.	4-10
4.2.4	Identification of Consistency Problems via RSL and RADX	4-12
4.2.5	Identification of Consistency Problems by Observation.	4-14
4.2.6	Identification of Problems by RADX	4-14
4.2.7	Summary of Deficiency Finding.	4-15
4.3	MAJOR MOM DFSR PROBLEMS.	4-23
4.3.1	Failure to Initiate Certain Batch Processing Output Reports.	4-24
4.3.2	Failure to Use the Parameter Report Controls	4-25
4.3.3	DABS Production and Use Inconsistencies. .	4-25
4.3.4	Work Order Number Character Field Confusion.	4-27
4.3.5	Lack of File Purge Instructions.	4-28
4.3.6	Missing File Contents.	4-28
4.3.7	Significant Quantities of Individually Minor Deficiencies	4-28
4.4	FINDINGS OF SREM PHASE RADX RUNS	4-30
4.5	REGENERATION OF REQUIREMENTS	4-31
5.0	A SYSTEMS ENGINEERING APPROACH TO THE EVALUATION OF REQUIREMENTS METHODOLOGIES	5-1
5.1	TECHNICAL COMPARISON OF SREM TO OTHER TECHNIQUES. .	5-2
5.2	THE SYSTEM ENGINEERING APPROACH FOR EVALUATION. . .	5-6
5.3	THE COST OF FIXING ERRORS VERSUS VERIFICATION COSTS	5-13
5.4	SOFTWARE REQUIREMENTS METHODOLOGY EVALUATION. . . .	5-15
5.5	CONCLUSIONS	5-18
6.0	ASSESSMENT AND RECOMMENDATIONS.	6-1
6.1	APPLICABILITY OF SREM TO ARMY MANAGEMENT INFORMATION SYSTEMS	6-1

TABLE OF CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
6.2	SREM APPLICATION TO TYPICAL MANAGEMENT INFORMATION SYSTEMS	6-3
6.2.1	CV-ASWM Application.	6-3
6.2.2	NAVDAC Application.	6-4
6.2.3	IHAWK/TSQ73	6-8
6.3	SREM ENHANCEMENTS	6-13
6.4	A COMPLEMENTARY TOOL FOR SREM	6-14
6.5	SUMMARY	6-18
6.6	RECOMMENDATIONS	6-20

TABLE OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1-1	The Seven Phases of SREM Engineering Activity	1-3
2-1	The Penalty of Requirements Errors.	2-2
2-2	An Elementary R_NET	2-6
2-3	Use of AND Nodes in R_NETs.	2-7
2-4	Use of OR Nodes in R_NETs	2-8
2-5	Sample R_NET Structure in Graphical and RSL Form.	2-14
2-6	REVS Functional Organization.	2-22
2-7	The SREM Methodology Process.	2-23
2-8	Relationship of Methodology Steps in Phase 1 of SREM. . .	2-28
2-9	Partial List of Prespecified RADX Tests	2-30
2-10	R_NET for the Engine Monitoring System.	2-31
2-11	Typical REVS-Produced Documentation	2-33
2-12	Typical CALCOMP Plot	2-34
2-13	Determination of Software Test Requirements	2-37
3-1	Interface Element Interrelationships.	3-8
3-2	Hierarchical Definition of the MESSAGE: WRK-ORD-REGISTRATION-DATA-MSG-IN.	3-10
3-3	Hierarchical Definition of the Output MESSAGE: FLOAT_STATUS_REPORT_MSG_OUT	3-13
3-4	Format for the Work Center Summary Report	3-14
3-5	Hierarchical Definition of the Output MESSAGE: WORK-CENTER_SUMMARY_MSG_OUT	3-15
3-6	Format for the Equipment Recall Schedule Report	3-17
3-7	Hierarchical Definition of the Output MESSAGE: EQUIP_RECALL_SCHEDULE_HEADER_MSG_OUT.	3-1
3-8	Hierarchical Definition of the Output MESSAGE: EQUIPMENT_RECALL_SCHEDULE_MSG_OUT	3-18
3-9	GLOBAL DATA and FILE Interrelationships	3-24

TABLE OF ILLUSTRATIONS (Continued)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3-10	Hierarchical Definition of the ENTITY_CLASS: CROSS_REFERENCE_FILE.	3-29
3-11	Examples of a SELECT and FOR EACH Node.	3-30
3-12	R_NET Interrelationships.	3-33
3-13	The Two Kinds of OR Nodes	3-35
3-14	R_NET, SUBNET Symbology	3-36
3-15	R_NET: PROCESS_MOM_KEYBOARD_INPUT (NET RT1000)	3-37
3-16	SUBNET: PROCESS_XMA_ENTRY (NET A1000).	3-38
3-17	SUBNET: PROCESS_XMA_A (NET A1001).	3-39
3-18	ALPHA Description Sheet for the SUBNET: PROCESS_XMA_A. .	3-42
3-19	RSL Listing of the Structure for SUBNET: PROCESS_XMA_A .	3-43
3-20	RSL Definition of the ALPHAs in SUBNET: PROCESS_XMA_A. .	3-44
3-21	Traceability Interrelationships	3-46
3-22	Excerpt for Chapter 4 of the DFSR Showing ORIGINATING REQUIREMENTS.	3-46
3-23	Hierarchy for INPUT INTERFACE	3-58
3-24	CALCOMP Plot of SUBNET: PROCESS_XMA_A.	3-61
3-25	Conditional Expressions for the CALCOMP Plot for SUBNET: PROCESS_XMA_A	3-62
4-1	AIRMICS Trouble Report Form	4-3
4-2	RSL Extensions to Support Trouble Report Entries.	4-5
4-3	RADX Checks for Incomplete Trouble Reports.	4-7
4-4	Decision Logic Table 319.	4-9
4-5	Distribution of MOM DFSR Deficiencies	4-17
4-6	Components of CATEGORY_OF_PROBLEM: Inconsistent.	4-18
4-7	Components of the CATEGORY_OF_PROBLEM: Ambiguous	4-19

TABLE OF ILLUSTRATIONS (Continued)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
4-8	Components of the CATEGORY_OF_PROBLEM: Missing	4-20
4-9	Components of the CATEGORY_OF_PROBLEM: Illogical	4-21
4-10	Components of the CATEGORY_OF_PROBLEM: Incomplete.	4-22
5-1	Methodology Evaluation Starting Point	5-7
5-2	Fault Tolerant Version.	5-8
5-3	Further Decompositions are Methodology/Tool Specific. . .	5-10
5-4	Cost of Requirements.	5-12
5-5	Example Analysis.	5-14
6-1	Illustration of Sequential MESSAGE Output Using Non-Terminating OUTPUT_INTERFACES	6-9
6-2	Event Logic Trees, The Basic for Automated Analysis . . .	6-17

LIST OF TABLES

<u>Table</u>	<u>Title</u>	<u>Page</u>
2.1	Current Nucleus of Defined RSL Elements, Relationships and Attributes.	2-15
2.2	MOM DSARC Extended RSL Elements, Relationships, and Attributes.	2-16
3.1	RSL Definitions Used in the Development of Interface Elements.	3-7
3.2	SUBSYSTEMs Identified in the MOM_DFSR	3-8
3.3	Comparison of Annex A Input Descriptions and Equivalent RSL MESSAGE Names	3-11
3.4	Comparison of Annex B Output Description and Equivalent RSL MESSAGE Names	3-19
3.5	RSL Definitions Used in the Development of ENTITY_CLASSES and ENTITY_TYPE	3-24
3.6	Comparison of Annex D Files and Equivalent RSL ENTITY_CLASSES and ENTITY_TYPES.	3-26
3.7	RSL Definitions Used in the Development of R_NETS	3-32
3.8	RSL Definitions Used in the Development of Traceability .	3-45
3.9	RADX Relational Operators	3-54
3.10	RADX Positive and Negative Connectors	3-55
3.11	Append_Item List.	3-60
3.12	SREM Phase 1 RADX Checks.	3-64
3.13	SREM Phase 2 RADX Checks.	3-64
3.14	SREM Phase 3 RADX Checks.	3-65
3.15	SREM Phase 4 RADX Checks.	3-65
3.16	SREM Phase 6 RADX Checks.	3-66
3.17	SREM Task/Time Allocation	3-68
4.1	Periodic Output Message Problem	4-26
4.2	Daily Accumulated Batch Storage (DABS) Inconsistencies. .	4-27
5.1	A Comparison of Some Requirements Techniques.	5-2

LIST OF TABLES (Continued)

<u>Table</u>	<u>Title</u>	<u>Page</u>
5.2	Relative Costs on Large Projects.	5-15
6.1	Description of Management Information Systems to Which SREM Has Been Applied	6-2
6.2	Survey of SREM Applicability to NAVDAC Applications . . .	6-6
6.3	SREM Applicability to the IHAWK/TSQ73 Application . . .	6-8
6.4	Enhancements Needed for Various SREM Applications to Systems with Characteristics Similar to Management Information Systems	6-12

1.0 INTRODUCTION

This document is the final report concerning TRW's demonstration of the application of the Software Requirements Engineering Methodology (SREM) to an existing Government Detailed Functional System Requirements (DFSR), Document under Contract DAHC26-80-C-0020, and is submitted in accordance with CDRL A002. It was prepared for the Army Institute for Research and Management Information Computer Science, located at Georgia Tech University, Atlanta, Georgia. It documents the result of TRW's application of SREM to TM 38-L71-2, Detailed Functional System Requirement (DFSR) - Volume IV, Standard Army Maintenance System (SAMS) - Retail Level, Maintenance Operations Management (MOM), (SAMS-1). The objective of this effort was to demonstrate the power of SREM as a tool to verify a software requirement, with the goal of determining the extent to which it was a complete, consistent, and unambiguous document. Specifically, the intent was to attain an understanding of what SREM is, how it is used, and what capability it possesses for isolating discrepancies in an existing software specification. In addition, this effort was intended to provide the basis for assessing the potential of SREM for inclusion as a tool in the current Army ADP-system development life cycle.

This report is published in two volumes. Volume I contains the basic report text, while Volume II contains the appendices that accompany this report.

1.1 SCOPE AND APPROACH OF THE SREM APPLICATION

SREM includes a seven-phase effort, which is shown pictorially in Figure 1-1. However, the SREM effort under this contract was limited to Phases 1 through 4. The efforts in the omitted phases involve the determination of VALIDATION_POINTS and VALIDATION_PATHs for PERFORMANCE_REQUIREMENTS plus two simulation phases for dynamically checking the software requirements.

The assessment of the adequacy of the MOM DFSR specification was accomplished utilizing the following tasks:

- Definition of interface elements
- Definition of stored information
- Definition of processing logic
- Definition of traceability
- Evaluation using the Requirements Analysis and Data Extraction (RADX) capability resident in the supporting software.

Additional tasks accomplished under the contract included a description of:

- The contents and application of SREM and its components
- The current state of SREM development
- The applicability of the current SREM approach to Management Information Systems
- SREM capability versus other similar techniques
- How SREM might be modified to improve its capability.

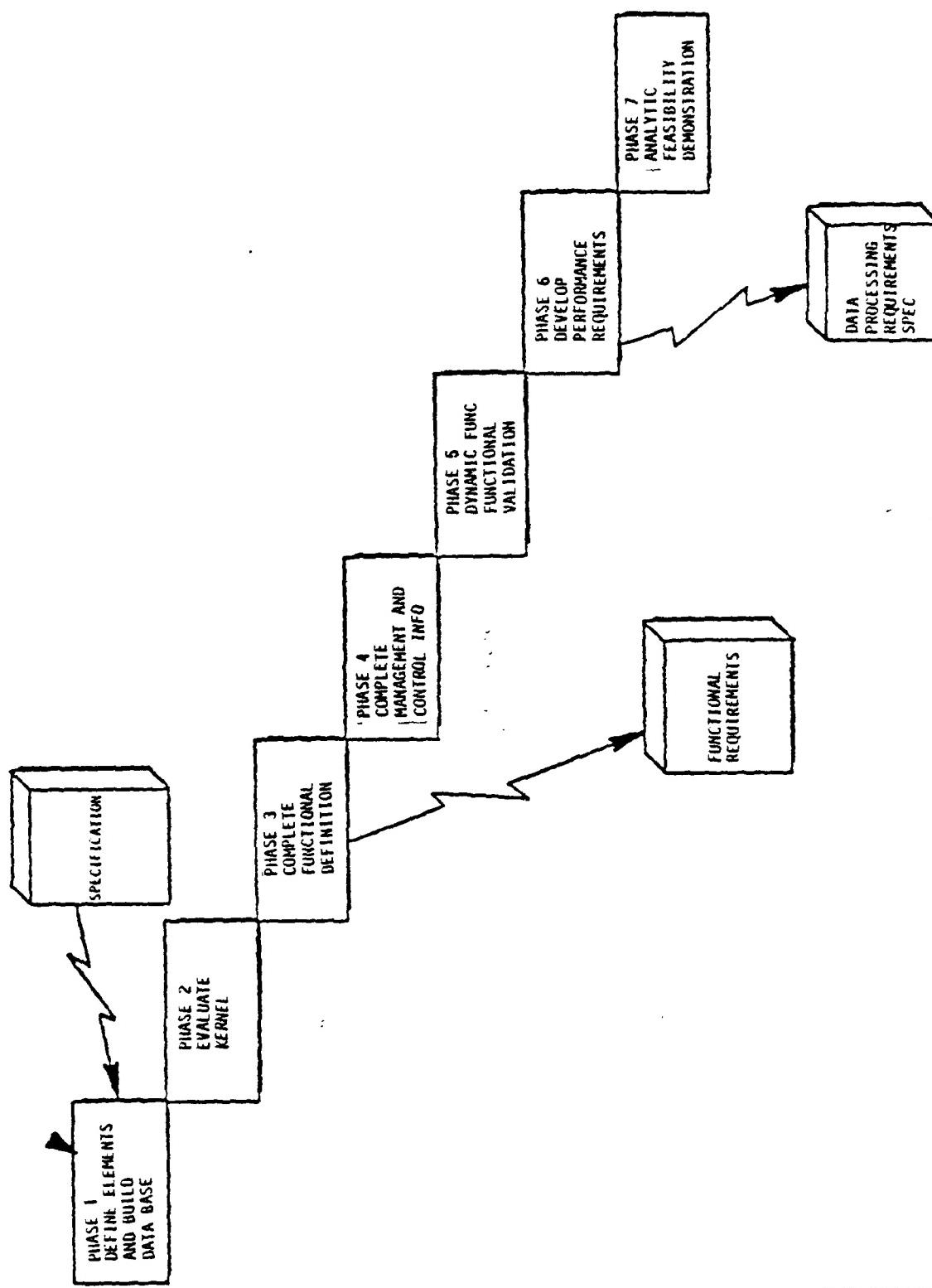


Figure 1-1 The Seven Phases of SREM Engineering Activity

AMX81-040

1.2 SUMMARY OF RESULTS

The power of the use of SREM to discover discrepancies in software specifications was confirmed as a result of this effort. A total of 302 Trouble Reports were written to document the discrepancies discovered in the logic of processing outlined in Decision Logic Tables (DLTs) within the MOM DFSR, and in their data consistency when compared to the data named in Annexes A, B, C and D. Although many of the discrepancies were minor, there were several significant problems, as are discussed in the body of the report.

No concerted attempt was made to determine all of the inconsistencies between the text descriptions, functional flow diagrams, and the DLTs. These internal inconsistencies were quite apparent, and a few have been reported via Trouble Reports. However, SREM was not designed to harmonize these kinds of problems. Rather, it is a tool to investigate the logic of the processing specified in the context of the inputs received and the expected outputs produced by the data processor. Accordingly, the SREM application was primarily applied to the most detailed available information concerning processing intent, as was provided by the DLTs.

Although there are some enhancements that should be investigated to aid the application of SREM for better support of Management Information Systems, SREM is clearly applicable to software applications of this type, and can be applied in its current state.

In addition to its proven capability as a verification tool for an existing software specification at the DFSR level, SREM can produce an even greater positive impact on software development if the DFSR-level specification is actually developed from the system level software requirements (presumably the General Functional Systems Requirement (GFSR)). When applied at that stage of development, the proposed approach can easily be communicated and understood. Because of the improved communication between the developer and the user early in the development cycle, the user can judge whether the proposed decomposition of the system level requirements (to the software requirements) will produce what is actually intended. If not, appropriate correction can be easily implemented well before such changes become prohibitively expensive. In addition, the effort by the software engineer to decompose the GFSR level requirements to the DFSR level using SREM will highlight areas where insufficient, ambiguous, or

conflicting information exists. Again, by raising such issues early in the development phase, the true intent of the user can be determined and factored into the DFSR very early in the development cycle.

A comparison of SREM to the capabilities of other software requirements techniques was accomplished. SREM was shown to be not only technically superior to the others, but to allow lower software application life-cycle costs as well.

1.3 OVERVIEW OF THIS REPORT

Section 2 of this report provides a tutorial description of SREM. This includes the following considerations:

- The Requirements Statement Language (RSL).
- The automated support tools incorporated in the Requirements Engineering and Validation System (REVS) with emphasis on the Requirements Analysis and Data Extraction (RADX) function, which provides the capability to test the adequacy of the requirements data base, and to provide documentation of its content.
- The application methodology, including development of Requirements Nets (R_NETs) used to define the processing response to the various input stimuli and to provide the ability for early software test planning.
- Management considerations.
- REVS software installation information.

Section 3 describes the basic tasks accomplished under this contract and their results. It includes a discussion of our approach to the definition of interface elements, stored information, processing logic (R_NETs), and traceability. Further discussion is provided with more detail on RADX use for evaluation of the requirements data base, and the results of the RADX evaluation are presented. Finally, a discussion of the statistics of the manloading application necessary to accomplish the SREM tasks is provided.

Section 4 discusses the results of the evaluation of the MOM DFSR. The kinds and degree of deficiencies found and documented by Trouble Reports are presented, and a discussion of the general effects of the DFSR deficiencies are summarized. A discussion on the regeneration of the DFSR using the documentation capability of RADX is also provided.

Section 5 compares SREM to other competing software requirements engineering tools, to include relevant similarities and differences. This discussion includes the important matter of total life cycle software costs as a criteria for selecting a requirements engineering technique. It is shown to be an appropriate way to assess the value of these tools, since more than just their initial application costs should be considered.

Section 6 completes this report and discusses the applicability of SREM to Management Information Systems. It also summarizes a few limita-

tions experienced with SREM in the verification of software applications, such as the MOM DFSR. Enhancement of RSL/REVS to ameliorate these limitations is discussed as a means to provide a more powerful SREM capability for developing and/or analyzing software requirements in the development of management information systems. All of these sections are contained in this volume (Volume I) of the report.

2.0 THE TRW SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY

The Software Requirements Engineering Methodology (SREM), developed for scientific systems, embodies four years of concentrated research directed toward the generation of better software requirements. As most of the fundamental concepts are directly applicable in the management information system environment, a re-statement of the original SREM objective and an overview of existing capabilities and recent developments is appropriate.

2.1 PURPOSE AND BACKGROUND OF SREM

In the Fall of 1974, the Data Processing Directorate of the U.S. Army Ballistic Missile Defense Advanced Technology Center (BMDATC) initiated a series of research programs directed to the development of a complete and unified approach to software development. These programs encompassed the total range of activities from development of system specifications through completion and testing of the software process design. Supporting programs were also conducted to perform basic research into such areas as software reliability, static and dynamic validation techniques, and adaptive control and learning.

A key element of the BMDATC programs was the Software Requirements Engineering Program (SREP). This was a research effort concerned with a systematic approach to the development of complete and validated software requirements specifications. As shown in Figure 2-1, errors made in the requirements phase become increasingly more expensive to locate and correct in the later phases of development.

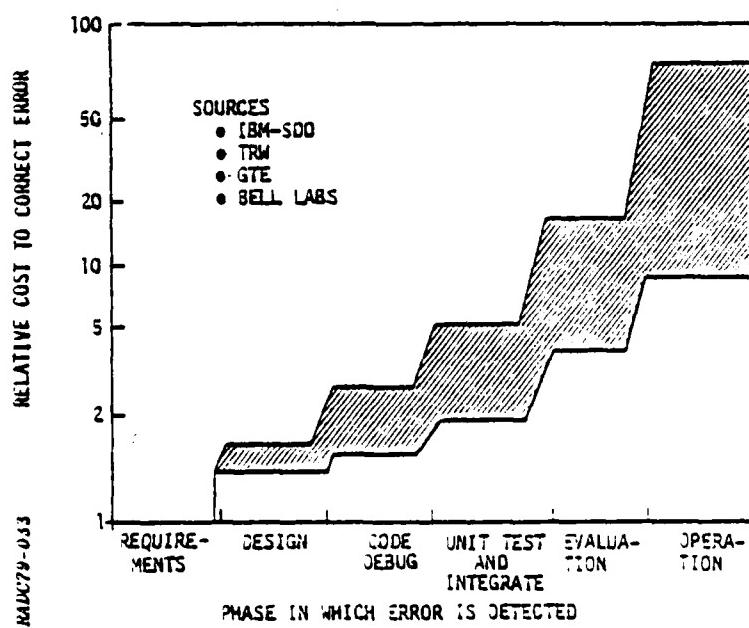


Figure 2-1 The Penalty of Requirements Errors

Ambiguity and lack of precision in the requirements statements lead to misinterpretation (and therefore errors) in the subsequent development phases and add further to cost and schedule overruns. Consistent with alleviating these problems with current requirements specification practices, the overall objectives of the SREP research were to:

- Ensure a well-defined technique for decomposition of system requirements into structured software requirements.
- Provide a mechanism for enhanced management visibility into the requirements development.
- Maintain requirements development independent of the target machine and the eventual software design.
- Allow for easy response to system requirements change.
- Provide for testable and easily validated software requirements.

2.2 SREM OVERVIEW

To meet these objectives, the Software Requirements Engineering Methodology (SREM) was developed. SREM is a formal, step-by-step process for defining data processing requirements. It provides a means to evaluate system requirements and enables preparation of good software specifications prior to design and coding.

SREM is designed to provide certain qualities often lacking in many software requirements. The most important of these are:

- INTERNAL CONSISTENCY, which is difficult to attain when applying traditional techniques on large systems.
- EXPLICITNESS, which requires unambiguous, complete descriptions of what is to be done, when, and with what kind of data.
- TESTABILITY, which ensures that ALL performance requirements are directly testable.
- TRACEABILITY, which allows easy impact assessment of changes to system requirements.

The basic elements of SREM are the methodology of application, its language, and the automated software tools to support the application effort. The Requirements Statement Language (RSL) provides the user with the ability to define software requirements in a form which assures unambiguous communication of explicit, testable requirements, and combines the readability of English with the rigor of a computer-readable language. The Requirements Engineering and Validation System (REVS) provides the automated tools for translating, storing, analyzing, simulating, and documenting requirements written in RSL. Through the use of RSL and REVS, the engineer can verify the completeness and consistency of a software specification with a high degree of confidence.

2.3 THE NEW PERSPECTIVE OF SREM

This methodology is an integrated, structured approach to requirements engineering activities. SREM begins with the translation and decomposition of system level requirements, or with the verification of an existing software specification. It performs analysis, definition, and validation of the ADP requirements; and ends with computer supported documentation of the completed software requirements. It represents a different approach and philosophy for software requirements engineering in that it embodies a flow orientation which eliminates many of the problems inherent in the classical functional hierarchy.

The common practice of organizing software requirements into a hierarchy of functions, subfunctions, etc., while superficially appealing, leads to difficulties in both the expression and the verification/validation of the requirements. This is due, in part, to the fact that such organization does not fit the basic input-process-output nature of data processing, and in part to the fact that a hierarchical tree of arbitrarily defined "functions" does not have a sufficiently rigorous mathematical basis to allow automated analysis for the completeness and consistency properties of the resulting specification. To avoid these difficulties, RSL and REVS are based on the concept of processing flow, a stimulus-response approach. Software requirements written in RSL are formulated in terms of a mathematical network (graph model) called a Requirements Network (R-NET). This approach provides several advantages:

- Describing the required processing in terms of a "logic diagram" of the system is natural to most engineers.
- The mathematical properties of an R-NET allow automated analysis for consistency and completeness through the application of graph theory.
- The flow orientation of an R-NET allows automated generation of simulation directly from the stated requirements.

Flows through a system are specified on the R-NETs, which consist of nodes which specify required processing operations and connecting arcs. The basic nodes are INPUT_INTERFACEs, OUTPUT_INTERFACEs, and required processing activities called ALPHAs. Through the use of these basic nodes, the required paths of processing can be specified. For example, if data is to be input to the data processor through an INPUT_INTERFACE called A,

processed by a processing step (ALPHA) called B, then processed by an ALPHA called C, and the result output through an OUTPUT_INTERFACE called D, then the required processing path can be specified by listing the sequence of operations:

INPUT_INTERFACE: A

ALPHA: B

ALPHA: C

OUTPUT_INTERFACE: D

This simple R_NET is illustrated graphically in Figure 2-2.

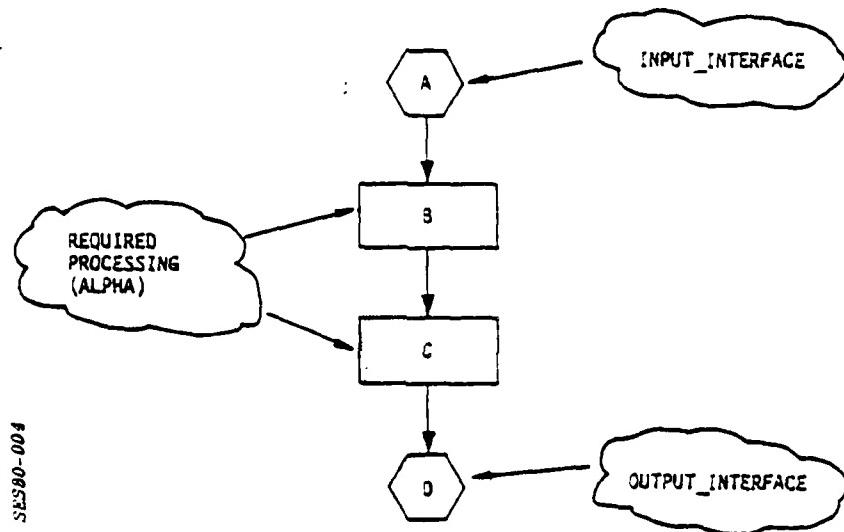


Figure 2-2 An Elementary R_NET

In the above example, the sequence ALPHA B-ALPHA C means that those processing steps must be performed in the indicated sequence. In many cases, the actual order of processing is immaterial. This can be specified through the use of an AND node as shown in Figure 2-3. This structure means that both B and C must be performed after receipt of data through A and before the result is output through D, but B and C are sequentially independent and may be performed in any order (or in parallel).

Most systems also require the specification of decision (control) points. Thus, in the above example, if B is to be performed under some circumstances (depending on the value of the input data, for example) and C is to be performed otherwise, a decision point and its attendant decision

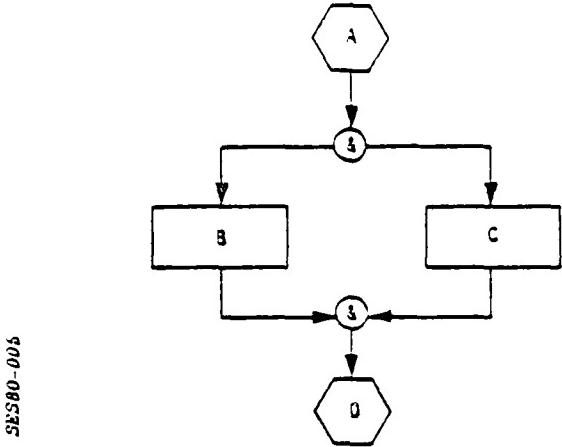


Figure 2-3 Use of AND Nodes in R_NETs

criterion must be specified. This is specified in an R_NET through the use of an OR node as illustrated in Figure 2-4. The second OR node following B and C means that the processing is to continue (i.e., output the results through D) if processing on either input branch has been completed.

Through the use of the INPUT_INTERFACE, ALPHA, and OUTPUT_INTERFACE, plus the AND and OR nodes, complete, complex processing requirements can be readily specified. Other nodes are provided to specify selection of data to be processed (SELECT, FOR EACH), to designate "test points" for specifying performance requirements (VALIDATION_POINTS), to provide internal controls (EVENTs), and to summarize detailed, subordinate processing flows (SUBNETs).

The Stimulus-Response approach, as discussed above, for the analysis and definition of software specifications has become the cornerstone of the SREM approach, and provides a new perspective for (as well as a concise means of) describing software requirements.

SKS80-006

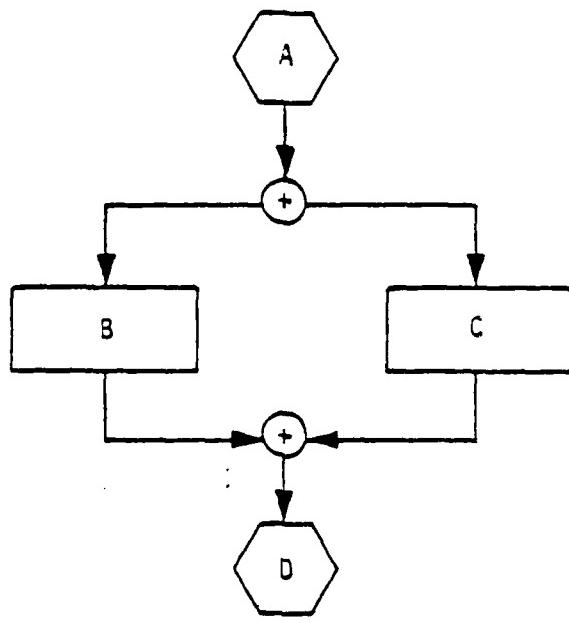


Figure 2-4 Use of OR Nodes in R_NETs

2.4 RANGE OF SREM ACTIVITIES

The starting point of SREM in the development of a software requirement from the system level is the point in systems engineering where the system requirements analysis has identified the functions and the stress points of the specified system; the interfaces between the subsystems (at least on the functional level); top-level system functions and operating rules (when and in what order functions are to be performed); and top level system functions have been allocated to the data processor. In the case of a verification effort, it begins with the determination to investigate the adequacy of an existing specification.

For both of the above cases, SREM is considered completed when the point is reached where primarily software development expertise is required to continue, the interfaces have been defined at the element level, all processing steps have been identified with appropriate DP requirements levied, all actions of the DP in response to a stimulus are determined in terms of sequences of processing steps, and the processing necessary to generate all required DP output interface messages has been specified. The basic difference for a verification effort is that deficiencies in the existing specification will have been documented and reported to the extent that the above characteristics are not present.

2.5 SREM DEVELOPMENT CONSIDERATIONS

The first step in defining the Software Requirements Engineering Methodology was to determine the properties required of a specification and of the individual requirements of which it is composed. The initial considerations were that:

- A specification is the set of all requirements which must be satisfied, and the identification of the subsets which must be met concurrently.
- A specification is neither realizable nor legally binding unless it is consistent with both the laws of logic and the laws of nature.
- A specification defines the properties required of a product, such that any delivery satisfying the specification satisfies the objectives of the specifier.

Taken together, the above considerations lead to a set of properties which a specification must have, from a technical point of view. They are:

- Internal consistency.
- Consistency with the physical universe.
- Freedom from ambiguity.

Economic and management considerations lead to the following set of properties which a good specification must exhibit:

- Clarity.
- Minimality.
- Predictability of specification development.
- Controllability of software development.

Since freedom from ambiguity was mandatory, a machine-readable statement of software requirements was defined. By employing an unambiguous language, and by translating and analyzing it with a program intolerant of ambiguity, a precise statement of requirements was ensured.

To provide an internally consistent specification, analyses of the requirements statements are performed by the REVS software. These analyses include semantic and syntactic decomposition of the individual statements, and analysis of the composite flow of data and processing. Support of consistency with the physical universe is accomplished by converting the

specification unambiguously into a simulation which can be executed against a model of the real world.

Recently, the Government has required that tactical software be developed in accordance with DoD Directive 5000.29. One key aspect of this requirement is that any software specification must be validated before being imposed. With the collection of tools and the methodology for their use, SREM provides a means for this validation through static and dynamic analysis at the requirements level.

Finally, to support control of both the specification process and software development, a means of selective documentation and analysis of the specification is provided. The integration of these tools with a sound and methodical engineering and management approach provides predictability in the specification process. Further, it aids in avoiding over-specification.

SREM was developed to ensure that software specifications express the real needs of the user. Although it was developed explicitly for high technology weapon systems problems, it is grounded on fundamental concepts relevant to all types of data processing.

2.6 PRINCIPAL COMPONENTS OF SREM

The three components of SREM are the Requirements Statement Language (RSL), the Requirements Engineering and Validation System (REVS), and the application methodology, itself. These components are described in this section.

2.6.1 The Requirements Statement Language (RSL)

The Requirements Statement Language is a user-oriented mechanism for specifying requirements. It is oriented heavily toward colloquial English, and uses nouns for elements and attributes, and transitive verbs for relationships. A complementary relationship uses the passive form of the verb. Both syntax and semantics echo English usage, so that many simple RSL sentences may be read as English with the same meaning. However, the precision of RSL enforced through machine translation, is not typical of colloquial speech. As a result, most complex RSL sentences are a somewhat stylized form of English.

The basic structure of RSL is very simple and is based on four primitive language concepts: elements, attributes, relationships, and structures.

Elements

Elements in RSL correspond roughly to nouns in English. They are those objects and ideas which the requirements analyst uses as building blocks for his description of the system requirements. Each element has a unique name and belongs to one of a number of classes called element types. Some examples of standard element types in RSL are ALPHA (the class of functional processing steps), DATA (the class of conceptual pieces of data necessary in the system), and R_NET (the class of processing flow specifications).

Attributes

Attributes are modifiers of elements, somewhat in the manner of adjectives in English, and they formalize important properties of the elements. Each attribute has associated with it a set of values which may be mnemonic names, numbers, or text strings. Each particular element may have only one of these values for any attribute. An example of an attribute is INITIAL_VALUE, which is applicable to elements of type DATA. It

has values which specify what the INITIAL_VALUE of the data item must be in the implemented software and for simulations.

Relationships

The relationship (or relation) in RSL may be compared with an English verb. More properly, it corresponds to the mathematical definition of a binary relation; that is, a statement of an association of some type between two elements. The RSL relationship is not symmetric; it has a subject element and an object element which are distinct. However, there exists a complementary relationship for each specified relationship which is the converse of that specified relationship. ALPHA INPUTS DATA is one of the relationships in RSL; the complementary relationship states that DATA is INPUT TO an ALPHA.

Structures

The final RSL primitive is the structure, the RSL representation of the flow graph. Two distinct types of structures have been identified. The first is the R_NET (or SUBNET) structure. As previously described, it identifies the flow through the functional processing steps (ALPHAs) and is used to specify the system response to various stimuli. The second structure type is the VALIDATION_PATH, which is used to specify performance of the system.

The goal of stabilizing requirements in a natural fashion using the stimulus-response approach, yet being rigorous enough for machine interpretation, was achieved primarily by orienting the design around the specification of flow graphs. Expression of structures in RSL is accomplished by mapping the two-dimensional graph onto a one-dimensional stream suitable for computer input.

The RSL structures are based on an extension to the basic theory of graph models of computation developed at UCLA to describe the intended operation of software. Many of the rules for constructing the RSL structures are fixed in order to enforce discipline, to preclude the user from forming flow patterns, and to ensure that each R_NET has a valid logical basis. Figure 2-5 shows the currently allowable nodes that may be incorporated into a legal network, and their equivalent RSL description.

Through the use of these four primitive language concepts, a basic requirements language is provided which includes concepts for specifying

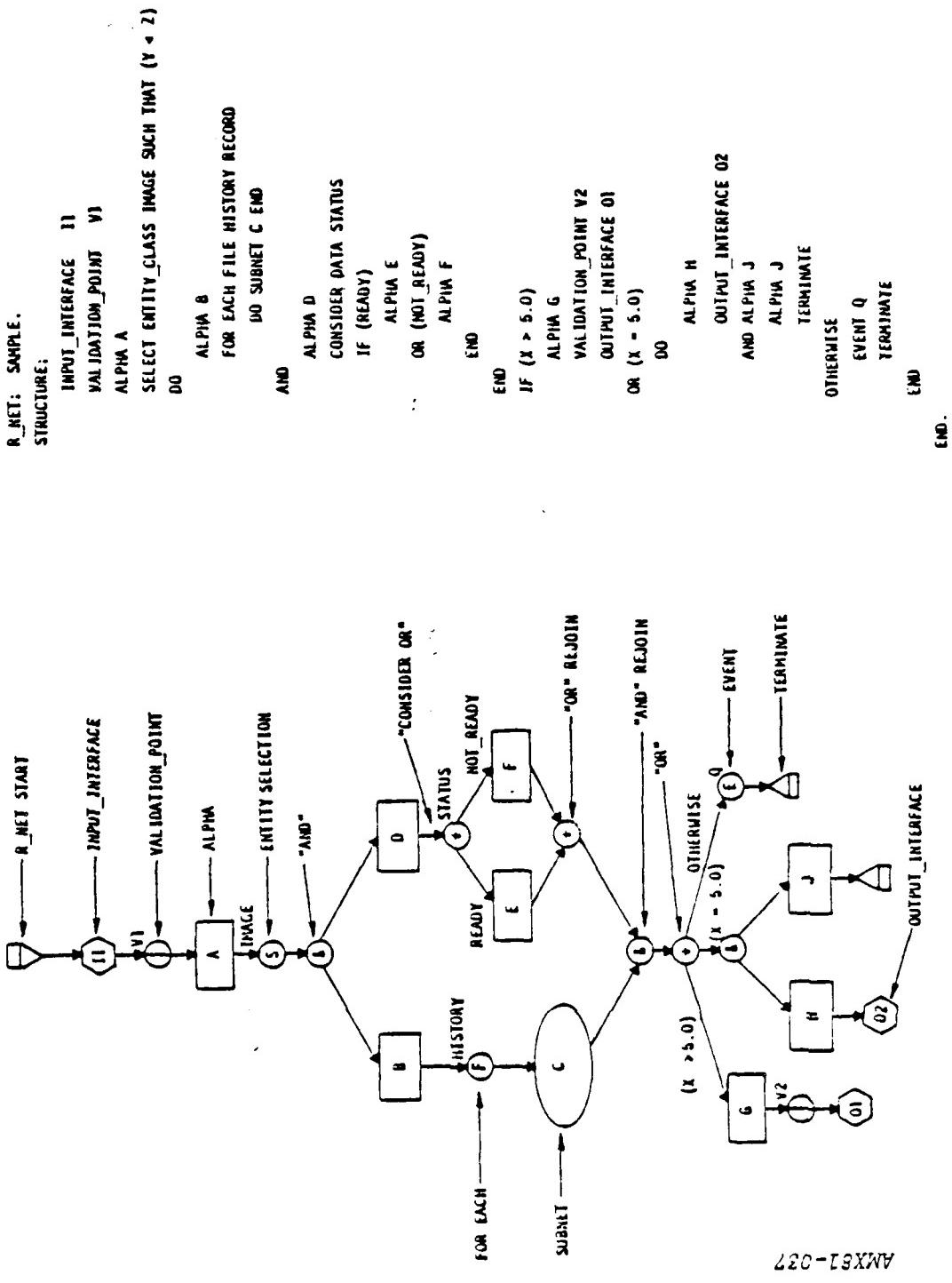


Figure 2-5 Sample R_NET Structure in Graphical and RSL Form

AMX61-037

processing flows, data processing actions, and timing and accuracy requirements. In addition, informative and descriptive material and management-support information may be specified. Using these primitives, a nucleus of concepts has been defined which, to date, has proven sufficient in scientific applications. Concepts supported by the current scientific version of the language are summarized in Table 2.1.

Table 2.1 Current Nucleus of Defined RSL Elements, Relationships and Attributes

ELEMENT TYPES	RELATIONSHIPS	ATTRIBUTES
ALPHA	ASSOCIATES	ALTERNATIVES
DATA	COMPOSES	ARTIFICIALITY
DECISION	CONNECTS	BETA
ENTITY_CLASS	CONSTRAINS	CHOICE
ENTITY_TYPE	CONTAINS	COMPLETENESS
EVENT	CREATES	DESCRIPTION
FILE	DELAYS	ENTERED_BY
INPUT_INTERFACE	DESTROYS	GAMMA
MESSAGE	DOCUMENTS	INITIAL_VALUE
ORIGINATING_REQUIREMENT	ENABLES	LOCALITY
OUTPUT_INTERFACE	EQUATES	MAXIMUM_TIME
PERFORMANCE_REQUIREMENT	FORMS	MAXIMUM_VALUE
R_NET	IMPLEMENTS	MINIMUM_TIME
SOURCE	INCLUDES	MINIMUM_VALUE
SUBNET	INCORPORATES	PROBLEM
SUBSYSTEM	INPUTS	RANGE
SYNONYM	MAKES	RESOLUTION
UNSTRUCTURED_REQUIREMENT	ORDERS	TEST
VALIDATION_PATH	OUTPUTS	TYPE
VALIDATION_POINT	PASSES	UNITS
VERSION	RECORDS	USE
	SETS	
	TRACES	

NAV79-030

RSL is an extensible language in that the primitives described above, which are initially built in, can be used to define additional complex language concepts. This extension capability of the language was exercised throughout the application of SREM to the MOM DFSR, and the extended concepts are shown in Table 2.2.

2.6.2 The Requirements Engineering and Validation System (REVS)

The Requirements Engineering and Validation System (REVS) is designed to allow the requirements engineer to state and modify requirements information over a period of time as the requirements are developed. The RSL

Table 2.2 MOM DSARC Extended RSL Elements, Relationships, and Attributes

ELEMENT TYPES	RELATIONSHIPS	ATTRIBUTES
DATA_ELEMENT	AS_OF	CARD_FIRST_COLUMN
LOGC_DCN	CODED_AS	CARD_LAST_COLUMN
MOM_FUNCTION	LOCATED_IN	DEFINITION
PREPARATION_DATE	SUPPORTS	FIELD_LENGTH
RELATIVE_POSN	USES	FIELD_TYPE
REVIEW_DATE		FREQUENCY_OF_USE
		GROWTH_RATE
		NORMAL_ACCESS_KEY
		NR_CHAR_PER_RECORD
		NR_CURRENT_RECORDS_PER_FILE
		NR_PROJECTED_RECORDS_PER_FILE
		PROPOSED_FILE_ORGN
		PROPOSED_MEDIA
		PURGE_RATE
		REQUIRED_ITEM
		RETENTION_PERIOD
		SECURITY_CLASSIFICATION

ANX81-072

statements that an engineer inputs to REVS are analyzed, and a representation of the information is put into a centralized requirements data base, called the Abstract System Semantic Model (ASSM). It bears this name because it maintains information about the required data processing system (RSL semantics) in an abstract, relational model. Once entered into the ASSM, the requirements are available for subsequent refinement, extraction, and analysis by the REVS software.

From a user point of view, there are five major functional capabilities which REVS provides.

- Processing of RSL.
- Interactive generation of Requirements Networks (R_NETs).
- Analysis of requirements and their listing in RSL and/or in specially formatted reports, using the Requirements Analysis and Data Extraction (RADX) capability.
- Generation and execution of functional and analytic simulators from functional requirements, models, or algorithms, and the generation and execution of

simulation post-processors from analytic performance requirements.

- Processing of extensions to RSL.

REVS and RSL allow the engineer to enter requirements into the requirements data base as they are developed, with REVS checking for consistency and completeness as new data is entered. Although the REVS capabilities may be applied in any order, generally, the user will initially build the requirements data base, and then request various Requirements Analysis and Data Extraction (RADX) static analyses to be performed. New entries will be made and additional RADX static analysis repeated until the requirements have been sufficiently developed for a simulation to be meaningful and useful. At that time, a simulator and post-processor may be generated. Once generated, it may then be executed numerous times and the data recorded and analyzed. Based on the results, this sequence may be repeated, starting with the modification of the existing requirements or the addition of new ones. The sequence will also be repeated as system requirements change, or when new requirements are imposed. When the user is satisfied that the requirements are correct, based upon the results of static and dynamic analysis, REVS will document the requirements in a form usable in a software requirements specification.

Each of the major capabilities identified above is allocated to a different functional component of REVS. The capabilities of these functions are described briefly below.

2.6.2.1 Processing of RSL

The analysis of RSL statements and the establishment of entries in the requirements data base corresponding to the meaning of the statements is performed by the RSL translation function of REVS. The translation function also processes the modifications and deletions from the data base commanded by RSL statements specifying changes to already-existing entries in the data base. For all types of input processing, the RSL translation function references the data base to do simple consistency checks on the input. This prevents disastrous errors such as the introduction of an element with the same name as a previously-existing element, or an instance of a relationship which is tied to an illegal type of element. Besides providing a measure of protection for the data base, this type of checking

catches some of the simple types of inconsistencies that are often found in requirements specifications at an early state, without restricting the order in which the user adds to, or alters, the data base.

2.6.2.2 Interactive Generation of R-Nets

Graphics capabilities to interactively input, modify, or display R_NET, SUBNET, and VALIDATION_PATH structures are provided through the REVS Interactive R_NET Generation (RNETGEN) function. RNETGEN permits entry of structures and referenced elements in a manner parallel to the RSL translator, and thus provides an alternative to the RSL translator for the specification of the flow portion of the requirements. Using this function, the user may develop (either automatically or under direct user control) a graphical representation of a structure previously entered in RSL. The user may work with either the graphical or RSL language representation of a structure; they are completely interchangeable.

The Interactive R-NET Generation facility contains full editing capabilities. The user may input a new structure, or he may modify one previously entered. When satisfied with the newly generated R_NET, the user may cause it to be stored, at which point it is automatically translated into an RSL representation and stored in the data base. At the conclusion of the editing session on an existing structure, the user may elect to replace the old structure with the modified one. The editing functions provide means to position, connect, and delete nodes, to move them, to disconnect them from other nodes, and to enter or change their associated names and commentary. The size of a structure is not limited by the screen size since zoom-in, zoom-out, and scroll functions are provided.

2.6.2.3 Analysis and Output of Requirements

The RADX function provides both static flow analysis capabilities and the capabilities of a generalized extractor system for checking the completeness and consistency of the requirements specification and for the development of requirements documentation. The static flow analysis deals with data flow through the R_Nets. The analysis uses the R_NET structure (in much the same manner that programming language data flow analyzers use the control flow of a program) to detect deficiencies in the flow of processing and data manipulation stated in the requirements. The generalized

extractor system allows the user to perform additional analysis and to extract information from the data base. The user can subset the elements in the data base based on some condition (or combination of conditions), and display the elements of the subset with any related information he selects.

Information to be retrieved is identified in terms of RSL concepts. For example, if the user wants a report listing all DATA elements which are not INPUT to any ALPHA (processing step), he enters the following commands:

SET A = DATA THAT IS NOT INPUT.

LIST A.

By combining sets in various ways, he can detect the absence or presence of data, trace references on the structures, and analyze inter-relationships established in the data base. In analyzing user requests and extracting information from the data base, the extractor system uses the definition of the language concepts, which are also contained in the data base. Thus, as RSL is extended, the extensions and their use in the requirements are available for extraction. Because of the importance of the use of RADX in the evaluation of the requirements data base and in the regeneration of requirements in this effort, a more detailed description of RADX is provided in Paragraph 3.7, preceding the description of our RADX evaluation effort.

2.6.2.4 Generation and Execution of Simulators and Post-Processors

The automatic Simulation Generation (SIMGEN) function in REVS takes the data base representation of the requirements of a data processing system and generates from the discrete event simulators of the system. These simulators are driven by externally generated stimuli. The baseline system generates simulators to be driven by a System Environment and Threat Simulation (SETS) type of driver program which models the system environment, (and the threat, where appropriate) and the components of the system external to the data processing system.

Two distinct types of simulators may be generated by REVS. The first uses functional models of the processing steps and may employ simplifications to simulate the required processing. This type of simulation serves as a means to validate the overall required flow of processing against higher level system requirements. The other type of simulator uses

analytic models. These are models that use algorithms similar to those which will appear in the software to perform complex computations. This type of simulation may be used to define a set of algorithms which have the desired accuracy and stability. Although real-time feasibility of a system cannot be established using this algorithm set, the simulation does provide proof of an analytic solution to the problem. Both types of simulations are used to check dynamic system interactions.

The SIMGEN function transforms the data base representation of the requirements into simulation code in the programming language PASCAL. The flow structure of each R_NET is used to develop a PASCAL procedure whose control flow implements that of the R_NET structure. Each processing step (ALPHA) on the R_NET becomes a call to a procedure consisting of the model or algorithm for that ALPHA. These models (or algorithms) for the ALPHA must have previously been written in PASCAL. The data definitions and structure for the simulation are synthesized from the requirements data element and their relationships and attributes in the data base.

By automatically generating simulators from the data base in this manner, the simulations are insured to match and trace to the requirements. Since all changes are made to the requirements statements themselves, new simulators can be readily generated as requirements change and are automatically reflected in the next generation of the simulator.

For analytic simulations, SIMGEN also generates simulation post-processors based on the statement of performance requirements in the data base. Data collected from an analytic simulation can be evaluated using the corresponding post-processor to test that the set of algorithms met the required accuracies.

Both REVS generated simulators and post-processors are accessed for execution through the Simulation Execution (SIMXCT) function for simulators, and the Simulation Data Analysis (SIMDA) function for simulation post-processors.

2.6.2.5 Processing Extensions to RSL

As mentioned earlier, the data base contains the RSL concepts used to express requirements, as well as the requirements themselves. Extensions and modifications to the concepts are processed by the RSL Extension (RSLXTND) function of REVS. The RSLXTND function is actually performed by

the same software as RSL translation, but is accessed separately to control extensions to the language through a lock mechanism built into the software.

2.6.2.6 REVS Organization

The above discussion has identified seven functions of REVS: RSL, RNETGEN, RADX, SIMGEN, SIMXQT, SIMDA, and RSLXTND. As shown in Figure 2-6, these functions are under the control of a higher level function, the REVS Executive. The Executive presents a unified interface between the user and the different REVS functions.

2.6.3 The Application Methodology

Historically, the methods for developing a software specification have been as numerous as the developers of such documents. In fact, few cases can be cited in which any formal methodology could be quoted. Until the specification appeared (often after thousands of man-years of effort), nothing was available to show that it would actually be generated. In addition, it has frequently been true that the quality of the specification, even with respect to elementary consistency from one requirement to another, could be verified only very late in software development. Since the problems were discovered only when the cost of correction was prohibitive, the requirements were frequently changed, degrading system performance in order to have a "workable" product.

In our research we found that a methodology was needed to guide the software development, and to make progress visible via measurable milestones. As shown in Figure 2-7, SREM starts with a specification which can be a formal specification, a conversation with the intended user, or a mental image of the system. In addition, it has been found that SREM may be applied to an existing software specification to verify its adequacy, as has been done in this effort on the MOMS DFSR.

The first step is to identify the specific functional and performance requirements of the system, and to record them in the requirements data base as ORIGINATING_REQUIREMENTS (what processing is to be done) or PERFORMANCE_REQUIREMENTS (how well must the processing be done; accuracy, timing, etc.). As the rest of the methodology proceeds, the need for the various elements defined and recorded in the data base (R_NETs, MESSAGEs,

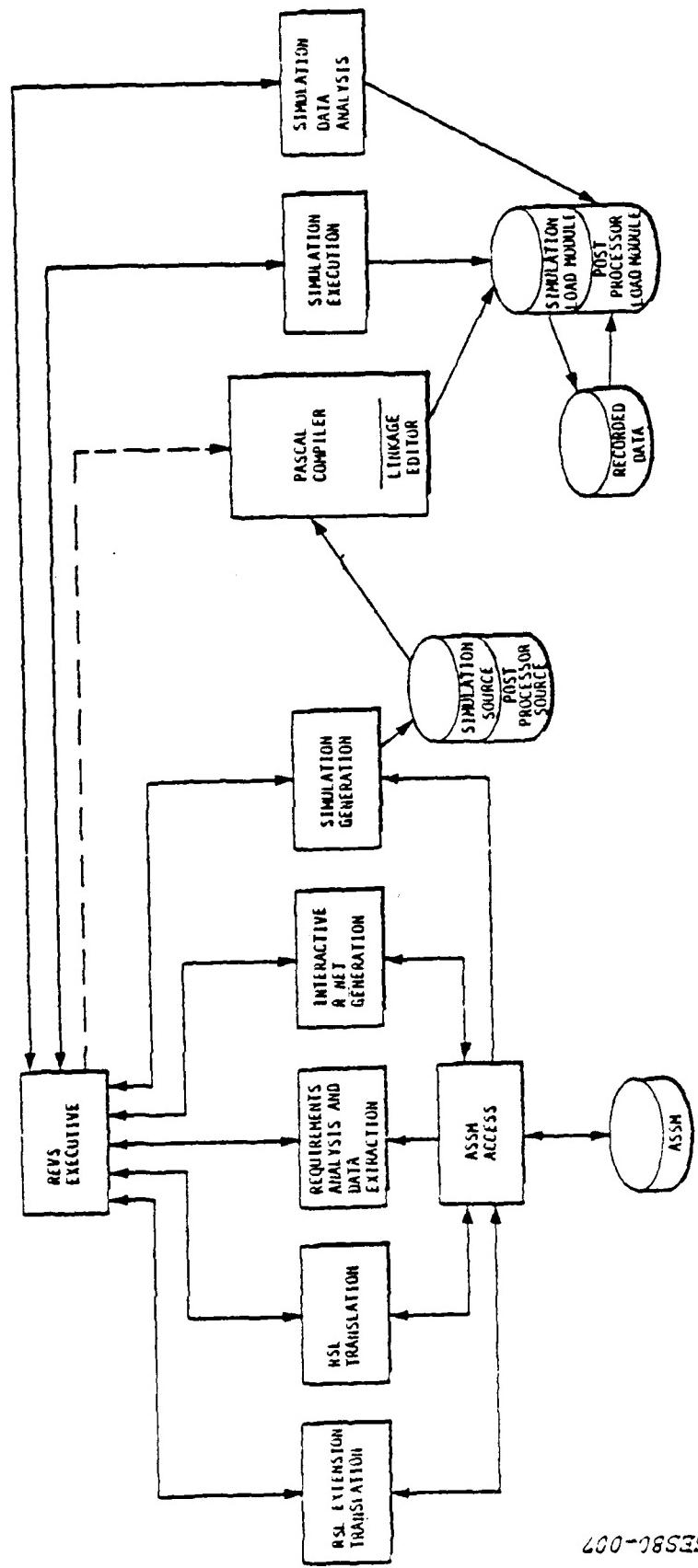


Figure 2-6 REVS Functional Organization

600-08535

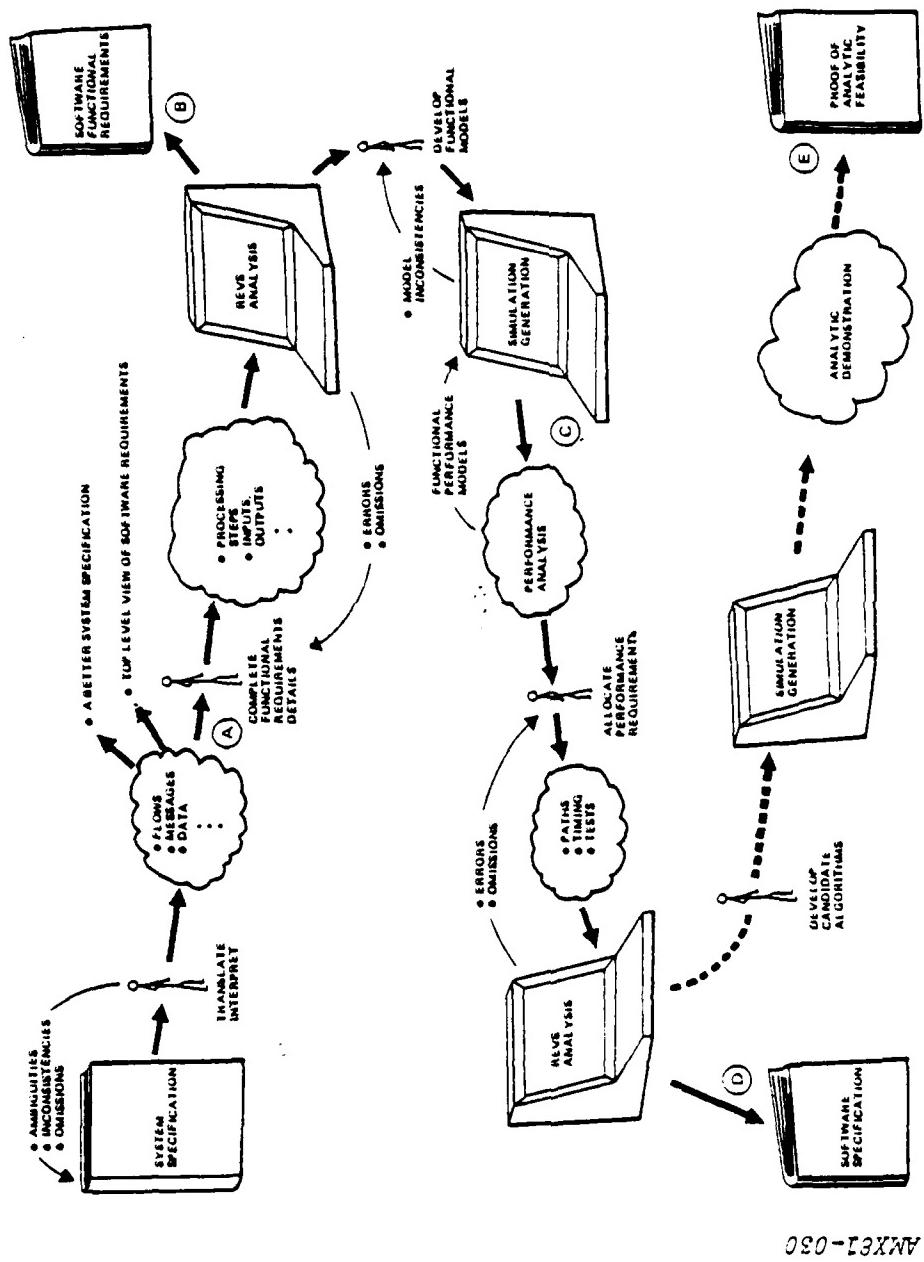


Figure 2-7 The SREM Methodology Process

AMXE1-030

etc.) should have stemmed from these ORIGINATING_REQUIREMENTS, and PERFORMANCE_REQUIREMENTS. Consequently, a traceability is established between pertinent elements and the requirement for them. Additionally, as ambiguous areas of the requirement are clarified, or as new guidance results, this fact can be documented in the data base as a DECISION, and various elements of the data base may then properly be traced to these decisions. With the requirements documented, actual definition of the software requirement can be initiated to meet the requirements.

Based on the system specification or software specification, all interfaces of the data processor are first identified, together with the MESSAGES that cross these interfaces and the MESSAGE contents. Next, DATA and FILE information is identified that defines the information required to be maintained about items of interest to the processing. These are stored in ENTITY_CLASSES or ENTITY_TYPES. R_NETs are then developed to specify stimulus/response relationships required of the software to process each of the input messages. We are now at Point A on Figure 2-7. During this process, specific problems in the system specification will be found (such as ambiguities and inconsistencies). These problems are corrected by an iterative process between the software engineer and the user until the specification is thought to be complete.

Next, the details of the functional requirements, including all of the input/output data relationships, the processing steps, the attributes, maximum values, minimum values, and the allowed data ranges, are completed. RSL is used to input these requirements into the data base, and RADX is executed to test for errors in consistency and completeness. When all the information has been input and all the errors corrected, the result is a functional specification (Point B), or in the case of a verification effort, a verified functional specification.

Before the functional specification can be finalized a simulation of the system is appropriate. First, simple functional models are developed for each of the processing steps and put through the simulation generation function to establish the model for simulation. Once all models are built, the simulator is executed to verify the entire process (Point C). Again, we emphasize that the simulation is actually accomplished using the requirements data base and the processing logic defined in the R_NETs. The result is confidence in the correctness of the requirements data base since

it provides checks not possible from static analysis alone. For example, the simulation can address whether the DATA input to the data base in one R_NET and used in a different R_NET is actually present for use when the using R_NET is exercised by the simulation.

After the functional requirement specification is validated, the PERFORMANCE_REQUIREMENTS are developed. Such PERFORMANCE_REQUIREMENTS usually aren't well structured; they are stated in system terms such as "kill probability" and "miss distances" for which the software shares only partial responsibility. The paths of the defined processing must be mapped and the paths which are being CONSTRAINED by these PERFORMANCE_REQUIREMENTS must be identified. Trade studies at this level will be performed until the right performance requirements are allocated for each path. When all of the paths are identified, the timing and analytic accuracy requirements are specified for each of the paths, and all of this information is input to the REVS data base. Completion of these steps results in a valid software specification (Point D).

Before attempting to build expensive real-time code it may be necessary to verify that the system is analytically feasible. To do this, one more simulation step, called the analytical feasibility demonstration, should be performed. This step will use real algorithms instead of functional models for the processing steps. It may not run in real-time, but it should consist of real algorithms expected to be used in the software. It should be driven by a driver with enough fidelity to represent the best understanding of the environment and measurements of the system to verify that the software will actually provide the accuracies required (Point E).

The methodology developed within SREM is not only formal (in that it provides an explicit sequence of steps leading to a validated specification) but it is also manageable, enumerating multiple phases for management review and analysis. Since it works from the highest levels of software definition (processing and data flows) to the most detailed (analytic models and data content) in a systematic manner, it supports early detection of high-level anomalies. A key feature of SREM is that the processing functions and data communications are considered in parallel, rather than have either following the other. As a result, the connectivity of the

system is always complete, and it becomes possible to partition the requirements effort among several groups early in the process without risking divergence, omissions, or inconsistencies. Some management aspects are covered in the following paragraphs.

2.7 SPECIFICATION MANAGEMENT

The management of a specification developed under SREM benefits most from the common source in the data base of all the representations of the software requirements. Thus, the simulation of the specification and the documentation of its requirements will be consistent at all times, since both have a single source of data for their generation without human intervention. In addition to a common data base, the methodology itself supports orderly software development planning and management, expedites review and analysis of the effort, assists communication of the details of the effort between participants, provides consistent rapidly attainable documentation, simplifies impact analysis of requirement changes, and assures early test planning capability. These valuable capabilities for any software manager are briefly illustrated below.

2.7.1 Management Planning and Control

Because the methodology is a step-by-step sequence of events with recognizable starting and ending points, it can be annotated with milestones, recorded on charts, and otherwise controlled with the management tools of the last several decades to provide predictability and control. This is not to suggest that the creativity of the specification process can either be scheduled or by-passed; it is still needed. However, the methodology isolates it into segments with high visibility, and supports management cognizance of its progress and impact.

Work may be easily assigned in a logical fashion because of the stimulus response approach of this methodology. Since each INPUT_INTERFACE identified for the system is the source of all the input MESSAGEs (stimuli) passing through it, a division of responsibility by INPUT_INTERFACE is appropriate. Thus, an engineer becomes responsible for developing the R_NET for that interface and all the appropriate related information for insertion into the requirements data base. Each requirements engineer generally accomplishes the same methodology steps with his assigned effort which, during the initial phase, are as illustrated in Figure 2-8. The typical Management Milestones are also shown in that figure.

It is not necessary for one step to be completely finished in all respects before the next can be undertaken. In most cases, the interfaces and MESSAGEs from some SUBSYSTEMs may be defined before those of others; in

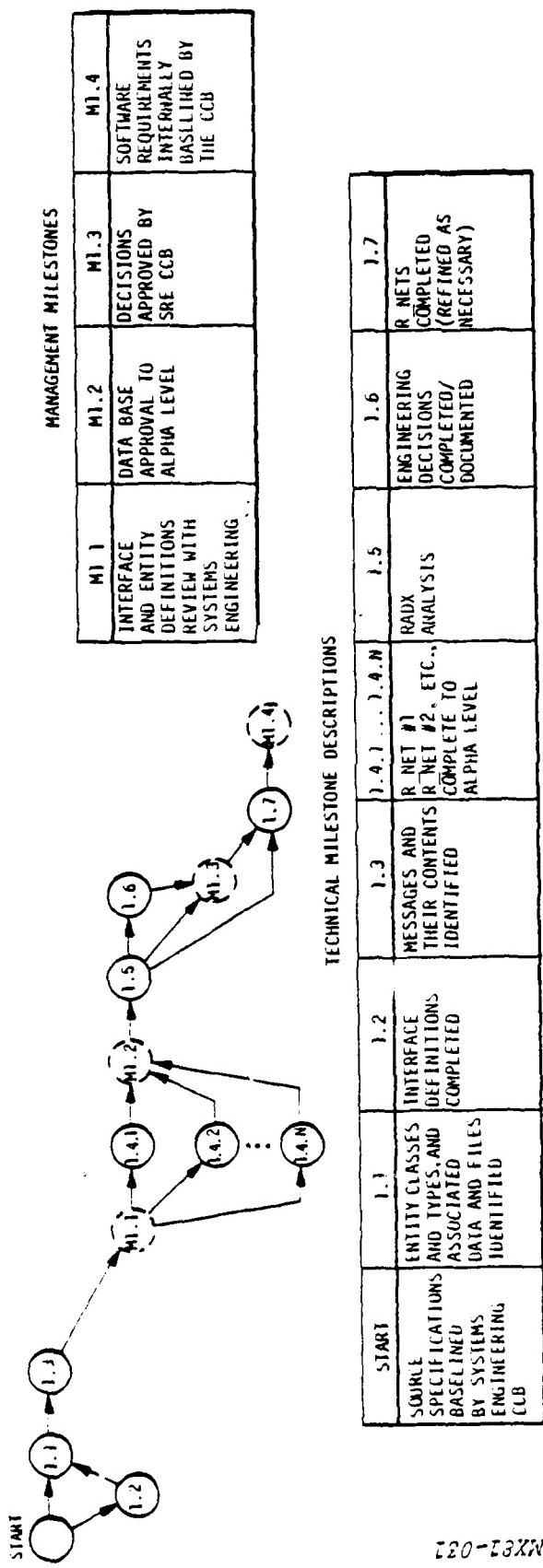


Figure 2-8 Relationship of Methodology Steps in Phase 1 of SREM

AMX61-032

this case, work could be initiated on the R_NET definition while interfaces are being defined for the remainder of the system. The parallel aspects of the network exemplify the modular capability for parallel R_NET development.

To be useful for management control, it must be possible to recognize when each milestone is completed. As will be shown, determination of the status of these and other milestones can be attained by querying the ASSM data base in various ways using the RADX system.

2.7.2 Review and Analysis

The software manager who is using SREM has a ready means to determine project progress (or lack thereof) for his internal review and analysis, or to meet the need for specific information to support reviews by higher level management, or by the user. This is accomplished through the capabilities of RADX. A standard set of RADX tests have been developed to assure that the data base will produce a complete, consistent, unambiguous specification. Figure 2-9 provides an example of a few of these pre-specified queries.

In addition, REVS users may form their own queries. At review time, for example, the manager can query the data base to determine the status of efforts expected to have been accomplished by the time of the review. This capability eliminates much of the need for subjective evaluation of progress. Here is a sample of the kinds of data that could be obtained:

- INPUT INTERFACEs that do not PASS MESSAGEs. This would indicate that MESSAGEs that will be passed by this interface have not yet been determined and entered into the data base.
- R NETs that are not ENABLED by an existing INPUT INTERFACE. This would indicate that the R NET for that INPUT_INTERFACE has not yet been defined in the data base.
- MESSAGEs that are not MADE BY DATA or FILE information. These are messages whose contents have not yet been defined.

These are only a few of the considerable list of queries that could be used to determine status. Actually, different sets of queries would be appropriate during each phase of the SREM process.

COMMANDS TO TEST SUBSYSTEM AND INTERFACE SPECIFICATIONS	
SET UNCONNECTED_SUBSYSTEM = SUBSYSTEM THAT IS NOT CONNECTED.	(*) ALL SUBSYSTEMS MUST BE CONNECTED.
SET INTERFACE = INPUT_INTERFACE OR OUTPUT_INTERFACE.	
SET INTERFACE_NOT_CONNECTED = INTERFACE WHICH CONNECTS	(*) TO AN INTERFACE THAT CONNECTS TO A SUBSYSTEM.
SET TOO_MANY_CONNECTS = INTERFACE THAT MULTIPLE CONNECTS	(*) TO AN INTERFACE THAT CANNOT CONNECT TO MORE THAN ONE SUBSYSTEM.
SET INTERFACE_W_MESSAGE = INTERFACE WITHOUT PASSES	(*) AN INTERFACE MUST PASS AT LEAST ONE MESSAGE.
SET OUT_MSG = MESSAGE THAT PASSED OUTPUT_INTERFACE.	
SET OUT_MSG_NOT_CONNECTED = OUT_MSG THAT IS NOT CONNECTED	(*) TO ALL MESSAGES THAT PASS TO OUTPUT_INTERFACE MUST BE FORWARDED.
SET MSG_NOT_PASSED = MESSAGE THAT IS NOT PASSED	(*) A MESSAGE MUST BE PASSED BY EITHER AN INPUT OR OUTPUT INTERFACE.
SET MULTI_PASSED_MESSAGE = MESSAGE THAT IS MULTIPLE PASSED	(*) A MESSAGE CAN ONLY PASS ONE INTERFACE.
SET MULTI_USAGE_INTERFACE = INPUT_INTERFACE THAT IS MULTIPLE REFERENCED	(*) TO AN INPUT_INTERFACE CANNOT BE REFERENCED BY MORE THAN ONE R_VAR.

COMMANDS TO TEST STRUCTURE SPECIFICATIONS	
SET INENABLED_NETS = A_NET THAT IS NOT ENABLED	(*) ALL NETS MUST BE ENABLED.
SET DEF_INPUT_INTERFACE = A_NET THAT REFERS TO INPUT_INTERFACE.	
SET MISSING_INPUT_ENABLE = REF_INPUT_INTERFACE THAT IS NOT ENABLED	BY INPUT_INTERFACE
	(*) AN R_VAR THAT REFERENCES AN INPUT_INTERFACE MUST BE ENABLED AT THE INTERFACE.
SET BAD_MULTI_ENABLE = REF_INPUT_INTERFACE THAT IS MULTIPLE ENABLED	(*) TO AN R_VAR WHICH REFERENCES AN INPUT_INTERFACE CAN ONLY BE ENABLED BY THE INTERFACE.
SET NOT_DEF_INPUT_INTERFACE = A_NET THAT IS NOT DEF_INPUT_INTERFACE	
SET BAD_INTERFACE_IS_ENABLEMENT = NOT_DEF_INPUT_INTERFACE THAT IS ENABLED AT INPUT_INTERFACE	(*) AN A_NET SHOULD NOT BE ENABLED AT AN INPUT_INTERFACE UNLESS THE INTERFACE APPEARS IN THE A_NET STRUCTURE.
SET STRUCTURE_NOTES = ALARM_SUMMARY, EVENT_VALIDATION_POINTS, INPUT_INTERFACE, OUTPUT_INTERFACE.	
SET NETS = A_NET IN SUMMIT.	
SET IMBALANCE_VALUES = STRUCTURE_VALUES SUCH THAT NOT REFERENCED TO AT NETS	(*) FOR THE REQUIREMENTS TO BE COMPLETED, ALL ALARM_SUMMARY, EVENT_VALIDATION_POINTS, INPUT_INTERFACE, AND OUTPUT_INTERFACE ELEMENTS MUST BE USED IN EITHER AN A_NET OR SUMMIT STRUCTURE.

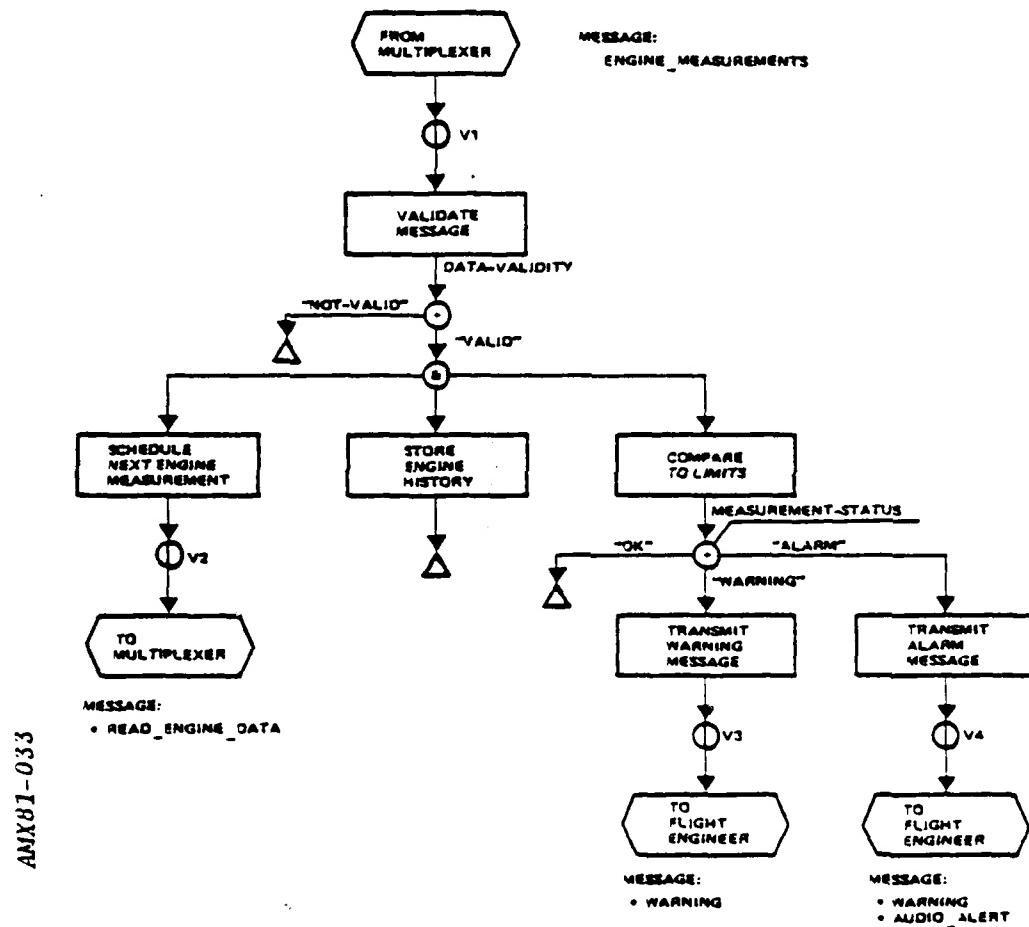
Figure 2-9 Partial List of Prespecified RADX Tests

The output from each query (called a SET) provides a number which identifies the quantity of elements in the data base that satisfy the specified query, and a listing of each of the elements in the SET. For example, in the third sample above, after indicating the quantity of MESSAGES that are not MADE_BY DATA or FILEs, the names of each MESSAGE would be listed. Because of the formality of RSL, and the specific meaning of its components, communication is facilitated in discussions with engineers on problems identified through RADX extractions such as these.

2.7.3 Technical Communication

Of all of the capabilities that SREM possesses, the capability to expedite technical communication is the one most often mentioned by users

as an unexpected payoff. The R_NET and SUBNET structures are the primary reason this is so. These structures are easy to understand and can distill large amounts of text into a compact flow diagram. The clarity of this approach is illustrated in Figure 2-10 for an R_NET describing the processing required for an Aircraft Engine Monitoring System. In this system, a



ANX81-033

Figure 2-10 R_NET for the Engine Monitoring System

MESSAGE is periodically received containing aircraft engine temperatures and pressures and is to be processed so that the flight engineer is warned

when engine conditions are out of limits. In addition, the engine readings are to be stored to provide engine history. Although the reader is probably not too familiar with R_NET structures, it is not very difficult to form a good idea of what is supposed to happen in the processing. The only concept not previously discussed is the circles labeled V1 through V4. These are VALIDATION POINTS which RECORDS DATA and FILE information needed for software testing.

Because of the ability the R_NET has for efficient communication, it is superb for use in engineering reviews, coordination between engineers, and discussions between managers and their engineers. In addition, and perhaps for the first time, the user can be given a chance to understandably review the perceived software requirements (via the R_NETs), with the distinct probability that he'll be able to identify areas where his intentions were not properly specified, and thus provide for correction before software design is started.

Another glance at the R_NET in Figure 2-10 will provide an understanding of the ease with which such errors can be identified. Note that at the OR Node immediately below the ALPHA called VALIDATE_MESSAGE, one of two branches is followed, depending on the result of the processing in this preceding ALPHA. If the result produces the DATA item called DATA_VALIDITY with the value of NOT_VALID, processing ends for that MESSAGE. An astute user would quickly perceive that if a long string of engine messages was producing a NOT_VALID value (perhaps due to some DP system failure), and even if the one of the engines was burning up, no warning would be given the flight engineer. This is clearly an unacceptable circumstance, and the requirements could be clarified so that the "NOT_VALID" branch was restructured to provide a warning after some number of consecutive invalid inputs.

2.7.4 Requirements Data Base Documentation

A further benefit for management is the capability of REVS to produce consistent specifications, even when many changes are made over short periods of time by many people in the project. One computer run yields the information needed for a complete specification with all the latest revisions. Of equal importance, if the proper thoroughness is applied to every revision, last minute panics do not introduce undetected errors in the documentation.

Figure 2-11 illustrates a typical printout showing the relationships in a hierarchical arrangement that presents a clear picture of some of the input considerations for the Engine Monitoring System. Output such as this is very useful for working-level documentation. It is compact, to the point, and readable, although it may not be sufficient for a formal specification.

LIST OF INPUT_INTERFACE DEFINITIONS

```
SUBSYSTEM : ENGINE_MULTIPLEXER
CONNECTED TO
INPUT_INTERFACE : MUX_INPUT
PASSES
MESSAGE : ENGINE_MEASUREMENTS
MADE BY
DATA : MEASUREMENTS
INCLUDES
  DATA : SENSOR_DATA
    INCLUDES
      DATA : MEASURED_P1
      DATA : MEASURED_P2
      DATA : MEASURED_P3
      DATA : MEASURED_T1
      DATA : MEASURED_T2
      DATA : MEASURED_T3
  DATA : SWITCH_DATA
    INCLUDES
      DATA : MEASURED_S1
      DATA : MEASURED_S2
```

AMX81-035

Figure 2-11 Typical REVS-Produced Documentation

REVS also provides the capability to produce the R_NETs via a CALCOMP plot. Figure 2-12 provides an example. Here again the plot incorporates all the latest data base changes and, therefore, is consistent with the specification printout. Together, the CALCOMP plot and the automated text output provide a total understanding of what the software is to do (Functional Requirements) and how well it is to do it (Performance Requirements).

SREM also can document problems of which the manager may want to remain cognizant. Usually at the early stage of system development, the specifications are incomplete, contain many ambiguities, and leave several

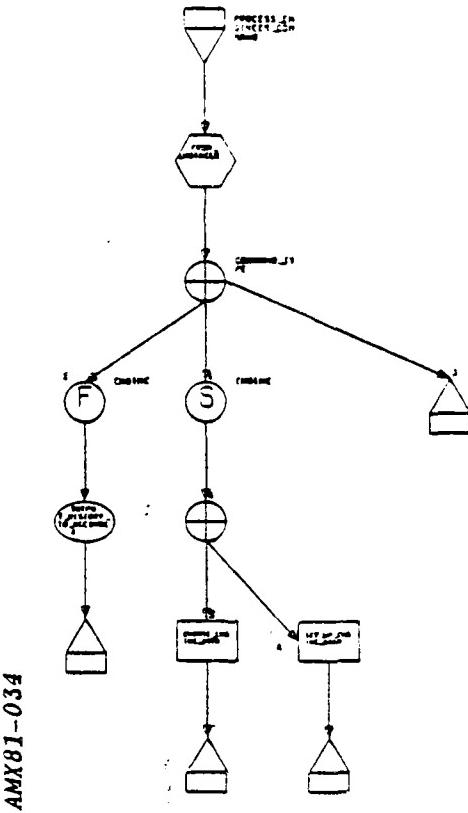


Figure 2-12 Typical CALCOMP Plot

issues for future resolution. SREM does not require that all omissions be resolved before proceeding. Instead, a requirements engineer can proceed from that which is clearly defined, and enter the requirements information into the data base.

Where the data is not immediately available, is inconsistent, or otherwise questionable, problem reports should be written and entered into the data base for follow up. For example, when R_NETs are defined, the requirements engineer will discover that:

- He doesn't know how certain MESSAGEs get processed under particular conditions.
- He doesn't know the conditions under which a certain output MESSAGE is to be produced.
- Certain MESSAGE contents are not known.

These situations and others like them can be recorded as problems by entering them as DECISIONs without CHOICE. These DECISIONs without CHOICE can be accessed via RADX query and printed out to present the manager with an automated problem list.

Once a CHOICE is established, usually via an answer from the user to a query, the answer is recorded in the data base as CHOICE. Thus, an up-to-date history of problems encountered, still open, and closed exists at all times during the SREM procedures. This not only provides a record of current status, but also a valuable record of the evolution of critical decisions on the project effort for future reference.

2.7.5 Assessing the Impact of Requirements Changes

As discussed earlier, a means is provided to establish traceability of requirements. ORIGINATING_REQUIREMENTS are DOCUMENTED BY paragraphs in SOURCE documents (system specifications, interface specifications, etc.) and each key element of requirements data base is traced back to one or more of these requirements. Although it requires some attention to detail for the initial establishment of this traceability, there is a real pay-off in assessing the impact of requirements changes. This impact assessment is especially useful during the configuration management procedures for a proposed change. The traditional, labor-intensive page-by-page search to attempt to identify impacts is so burdensome that it is seldom completely successful. As a result, there is concern that removing some portion of the software may impact the processing in unforeseen ways.

By taking the time initially to establish traceability using the SREM process, impact assessments of requirements changes are greatly eased. For example, the following query could be formed and entered via RADX to determine the impact of a change in the ORIGINATING_REQUIREMENT named PROCESS_RADAR_INPUTS.

SET: ANY ELEMENT = DATA, FILE, MESSAGE, INPUT_INTERFACE,
OUTPUT_INTERFACE, SUBSYSTEM, R_NET, SUBNET.

(Creates a set of all the indicated RSL elements)

SET: IMPACTED ELEMENTS = ANY ELEMENT WHICH IS TRACED FROM
ORIGINATING_REQUIREMENT: PROCESS_RADAR_INPUTS.

(Creates a set of all elements named in the first set which trace to the indicated ORIGINATING REQUIREMENT. These are the elements that could be impacted by this change.)

LIST: IMPACTED_ELEMENTS.

(This would result in a printout listing all elements that are impacted by the change; that is, all those in the derived set called "IMPACTED_ELEMENTS".)

Thus, with a single computer run, all portions of the previous requirements engineering effort are identified that should be re-examined to determine changes or realignment needed to adjust to the new requirement change. More important, however, is that all the relationships these elements have with other unchanged requirements is also provided by the run. This assures that there is a full understanding of what areas would be impacted by deletions or changes being contemplated. A comparison of this approach to typical manual reviews yields a real appreciation of the power of this capability.

2.7.6 Software Test Planning

Software test planning difficulties are reduced by the SREM Analysis approach of identifying processing paths (R_NETs) for various input stimulus (MESSAGES). It impacts both the activities to be performed, and the techniques for planning and managing test activities.

The definition of all possible branches of processing, to include error paths, is a natural result of defining the R_NET for each input stimulus (MESSAGE). Thus, each resulting branch is a possible candidate for a software test. Generally, however, a smaller set of software sets are designed by concentrating on the paths of greatest importance. A VALIDATION_POINT may be added to such branches to indicate the appropriate test data recording points for PERFORMANCE_REQUIREMENTS in the process.

Figure 2-13 illustrates two methods of determining PERFORMANCE_REQUIREMENTS for various R_NET branches. As shown on the left, such performance may be derived by allocation from the performance described in the system requirements. In addition, there may be a one-on-one application of a PERFORMANCE_REQUIREMENT to an R_NET branch, as shown on the right. In either case, each PERFORMANCE_REQUIREMENT is linked to (CONSTRAINS) a

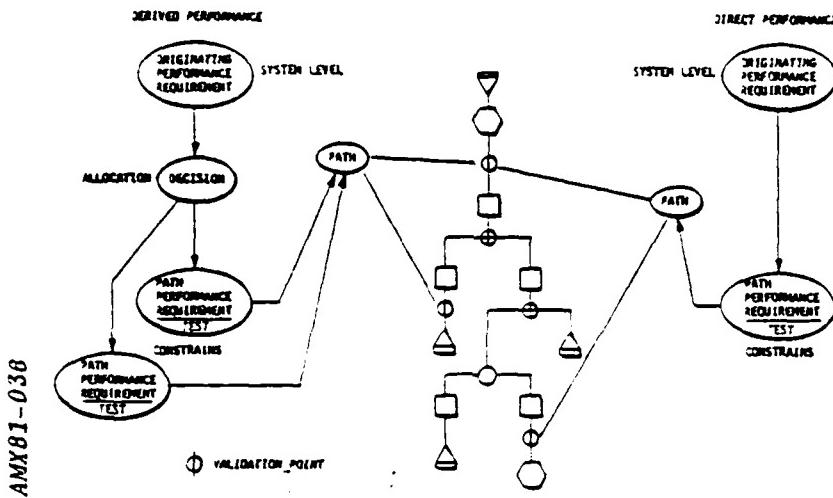


Figure 2-13 Determination of Software Test Requirements

VALIDATION_PATH, which includes VALIDATION_POINTS for measuring the test data needed to determine that the PERFORMANCE_REQUIREMENT has been met.

A VALIDATION_POINT is analogous to a test point in a piece of electronic hardware. It is a port through which information is collected in order to assess performance of the function under test. The information RECORDED_BY a VALIDATION_POINT may be DATA or FILEs. A PERFORMANCE_REQUIREMENT has an attribute called TEST, which uses the DATA and FILE information RECORDED_BY the VALIDATION_POINTS on a VALIDATION_PATH to determine the pass/fail criteria for the TEST. All the information concerning PERFORMANCE_REQUIREMENTS, VALIDATION_POINTS, VALIDATION_PATHs, and TESTs are entered into the centralized data base.

This process assures that needed tests are derived and documented for all appropriate processing paths, and that needed test data will be known for each test. The positive implications for this on the attainment of a testable software specification is clear and significant.

Two additional points are appropriate:

- In order for the tester to determine the tests appropriate to allow proper verification that the software requirements are met, he has to discover all the stimulus-response paths of consequence in the system. No matter how the requirements are generated, the tester is faced with this task. Thus, it makes

sense to develop the requirements using the same stimulus-response approach, and thereby to avoid double analysis of the requirement.

- The identification of testing is very early in the software development because it occurs during the requirements phase. Since it is done in a way that is directly usable by the tester, there is real economy of effort and early confidence of appropriate testing by all concerned.

2.8 MAINFRAME REV'S SOFTWARE CONFIGURATIONS

The latest version of REV'S, Version 14, is now installed and operational at the following sites:

- BMDATC Advanced Research Center (ARC), Huntsville, Alabama.
- TRW Defense and Space Systems Group (DSSG), Redondo Beach, California.
- Naval Air Development Center (NADC), Warminster, Pennsylvania.

This section describes the REV'S configuration at each of these sites and provides current data as to the size of the REV'S software, in terms of lines of source code and execution memory requirements.

2.8.1 REV'S Source

The basic REV'S program consists of approximately 43,000 lines of PASCAL organized into 1108 procedures and 291 lines of FORTRAN in 8 routines. At NADC, an additional 110 lines of FORTRAN in 5 routines is required to support CALCOMP plotting. The REV'S PASCAL source by functions is as follows:

<u>Function</u>	<u>Source Lines</u>
Executive	6921
CALCOMP Plotting	1072
RADX	9665
RNETGEN	4031
TESTER	2900
Translator	7587
SIMDA	492
SIMGEN	9583
SIMXQT	622

2.8.2 TRW PASCAL Development System

The PASCAL development system is used to support installation, maintenance, and execution of REV'S consisting of a PASCAL compiler, run-time

library, and several utility programs. The compiler itself consists of 7503 lines of PASCAL in 143 procedures. The run-time library consists of 43 routines with 1055 lines of PASCAL and 2235 lines of COMPASS. The 5 utility programs consist of 2294 lines of PASCAL.

2.8.3 Data Base Control System (DBCS)

The DBCS consists of 187 routines with a total of 10,520 lines. All of these are written in FORTRAN except for one COMPASS routine of 141 lines. The inputs to the DBCS necessary to define the ASSM structure consist of 195 card images. An additional 47 card images are required to define the structure of a REVS post-processor data base.

2.8.4 Compiler Writing System (CWS)

The CWS consists of 5 PASCAL programs containing 5562 lines and 3 standard input files totaling 946 lines of PASCAL. The definition of RSL consists of a total of 7195 source lines in 3 files. This source is a mixture of PASCAL code and a syntactic and semantic definition of RSL.

2.8.5 Ancillary Files

To construct and support REVS-generated simulators and post-processors, several additional files are defined. The RISF, an input file to SIMGEN, consists of 1313 PASCAL source lines. The post-processor data base builder program and post-processor run-time library consists of 137 lines of FORTRAN in 5 routines and 454 lines of PASCAL in 23 procedures. The RSL translator input statements necessary to define the nucleus data base are contained in a file of 804 card images.

2.8.6 BMDATC ARC Installation

At the ARC in Huntsville, Alabama, REVS is installed on a CDC 7600 operating under SCOPE 2.1.5. The REVS program is organized into 42 segments as defined by 137 input directives to the Segment Loader. The execution of REVS is controlled by a 764-line COMPASS program which emulates the macros defined for REVS.

This installation of REVS provides for 200 639-word data base pages in large core memory (371470_8 words of LCM) and loads in a field length of 117116_8 SCM and 377040_8 LCM. A nominal REVS execution requires 160000_8

words of SCM. All REVS capabilities, including interactive ANAGRAPH access, are available at the ARC.

2.8.7 TRW DSSG Installation

REVS is operational on TRW/TSS (TRW computer center in Redondo Beach, California) for batch and remote batch use. REVS operates under the MACE operating system on the CDC CYBER 70/74 and CYBER 170/174 computers. The REVS program is organized into 35 overlays as defined by 125 input directives to the TSS loader. The execution of REVS is controlled by six control-card PERFORM files containing 119 card images. A utility program consisting of one 4-line FORTRAN routine and a 53-line PASCAL procedure is defined to generate file identification banner pages.

Two REVS configurations exist on TSS. One allows 64 639-word data base pages in central memory (117700_8 words of CM), the other allows 100 data base pages in central memory (174634_8 words of CM). The two absolute programs required 226236_8 and 303302_8 words of CM loads. Nominal execution field length requirements are 250000_8 and 325000_8 , respectively.

2.8.8 NADC Installation

REVS operates on the two CDC 6600s and the CYBER 170/175 at NADC (Warminster, Pennsylvania) under control of the KRONOS 2.1.1 operating system. The REVS program may be executed in a batch or remote-batch mode and is organized into 45 segments as defined by 357 input directives to the CYBER Loader. The execution of REVS is controlled by an 873-line COMPASS program which emulates the job central macros defined for REVS.

Three distinct versions of REVS are configured for the three computers, each with a different ECS data base page buffer size (200, 400, or 500 256-word pages). The memory requirements on the three machines are as follows:

	Machine	Machine	Machine
	A	B	C
ECS	310000_8	144000_8	372000_8
CM Load	105022_8	73570_8	105332_8
Nominal CM to execute	150000_8	130000_8	150000_8

2.9 REVS IMPLEMENTATION ON THE DEC VAX 11/780

As part of a contractual effort from the Ballistic Missile Defense Advanced Technology Center (BMDATC), TRW has installed REVS on a VAX 11/780. The reasons for this implementation were that:

- It would be applicable for the development of the Architecture Development Language (ADL) effort, which allows the description of arbitrary architectures of proposed computer system constructs. These are the basis for modeling the system for the support of investigation of the use of large numbers of interconnected microprocessors on the BMDATC Advanced Testbed as DP subsystem candidates.
- With modification, it may have application to creation of testbed software.
- It will support DDP application investigations.
- It will allow many new users to have REVS for their use (where now, there is limited availability on a few CDC machines) such that broader SREM application is possible.

As part of this effort, TRW was to compare CPU times between the VAX and the CDC 7600, to determine where the VAX CPU time was accumulating during the execution of the REVS program, and to assess how these run times could be reduced. The ratio of CPU times varied between 50 to 1 and 400 to 1, with test cases averaging 128 to 1. The heaviest usage was during RADX executions, and since most of the test cases involved RADX, there was a bias in the run times ratio toward the upper extremes. Because run time differences of about 20 to 1 were expected, research was needed to investigate the reasons, and was an important part of the study.

As a result of efforts under the BMDATC study, improvements, amounting to about 50 percent reduction in the above run time differences were attained. Further reductions will require the optimization of the data base management system (optimization was not a part of the study effort). The results of the BMDATC effort have been excerpted from the final report and printed in Appendix A for those desiring added information.

3.0 DESCRIPTION OF THE SREM APPLICATION TO THE MOM DFSR

3.1 INTRODUCTORY REMARKS

The purpose of this section is to present a profile of our efforts to apply SREM to the DFSR. After this introduction, and a discussion of the scope of the effort, a description of the SREM process will be presented to describe our definition of:

- Interface elements.
- Stored data.
- R_NETs.
- Traceability.

In each of these descriptions, the appropriate RSL concepts will be introduced, together with a description and illustration of the approach to the definition process. A comparison of the resulting RSL elements to those described in the DFSR will then be provided followed by a discussion of problems encountered.

Following that, and in preparation for the discussion of our evaluation of the requirements data base, a tutorial on RADX and its use for this evaluation will be presented. This will be followed by the results of our RADX evaluation.

Finally, an analysis will be presented concerning how the time of the software engineers assigned to this effort was applied. This will be based on the diaries maintained during the MOM DFSR evaluation. The analysis will include statistics of interest concerning the amount of effort needed for evaluation of a specification of this size, and how it was distributed to the various phases.

3.2 SCOPE OF THIS EFFORT

The evaluation of the DFSR under this contract was constrained by several factors encountered during the period of performance. The size of the specification package was significant for the time available and the level of effort possible under the contract. In addition, the specifications possessed two unexpected conditions:

- There was a significant quantity of errors, beyond the number that was anticipated.
- There was an unexpectedly large quantity of input MESSAGES to be considered, since each individual data item was actually a separate input for which processing was defined.

3.2.1 Approach to Overcoming Delays Caused by the High Error Count

We found such significant inconsistencies between the text which described the required processing, the functional flowcharts, and the Decision Logic Tables (DLTs) that we failed when we first attempted to harmonize these three descriptions in order to synthesize the intent of each portion of the processing. Complicating this problem was the fact that no user existed whom we could query to determine the true intent. Rather, we were required to provide our own answers. This turned out to be so time consuming that we had to find a different approach if there was to be any chance of completing the effort in the time available.

As a result, it was clear that we would have to pick one of the three conflicting sets of processing descriptions and use it in isolation from the others in applying the SREM methodology. After evaluation of each of the possible sources, we selected the Decision Logic Tables (DLTs) for use, primarily because they presented the most complete processing description of the three.

In selecting the DLTs, we consciously decided to ignore the inconsistencies between text, functional flow diagrams, and DLTs. Time did not permit such an evaluation, although a few differences that were uncovered during DLT evaluation were documented in Trouble Reports. We are satisfied that many added Trouble Reports would have resulted if we had evaluated text and functional flow diagram consistency as carefully as we did the DLTs. These ignored inconsistencies would only have reflected format content inconsistencies within the written specification, rather than the

more important adequacy and internal consistency of the processing logic. Thus, our attention was focused within and between the DLTs, and between the DLT definition of data identification compared to that within the following DFSR Annexes:

- Annex A: Input Descriptions.
- Annex B: Output Descriptions.
- Annex C: Information Elements.
- Annex D: File Description.

However, the decision to concentrate on DLTs only partially solved the size of the problem. When we followed the approach of analyzing the DLTs, the error count still turned out to be beyond that originally expected, and considerably more time was necessary for Trouble Report preparation and coordination than had been contemplated. As a result, further constraints on the effort were found to be needed so as to allow completion within the allotted schedule. At this point, it was determined that the goals of this effort would be duplicated if SREM was applied to both the MOM and MPOM DFSRs. The format and contents of the MPOM DFSR was similar to that of the MOM. It was however, smaller and less complex, since the man-machine, real-time aspect of the MOM specification was missing, and since there were fewer input and output messages and global files to consider. Consequently, since the MOM DFSR was more varied, and since it contained far more processing requirements than the MPOM DFSR, it was decided to concentrate on the MOM DFSR for our SREM application demonstration.

3.2.2 Approach to Overcoming the High Input MESSAGE Count

THE SREM methodology was designed to treat each group of input information received or transmitted by the DP as a MESSAGE. Each input MESSAGE provides a processing stimulus, the response to which is defined on an R_Net. In the case of the MOM DFSR real-time processing, each of several hundred data items is input to the system in response to a prompt cue provided by the DP to the operator after he has successfully input the preceding data item in the proper format and containing a legal value.

In terms of the current methodology, each DATA item thus input actually is a separate RSL MESSAGE and presented the need to define a processing path for each one. However, the SREM process contemplates the logical processing of a MESSAGE, and not necessarily the detailed means for implementing the correct construction of these MESSAGES as defined for individual data inputs. It was clear that the desired output products of this effort could not be attained if each such data input was treated as an individual input MESSAGE. A way had to be found to shortcut this problem without adversely impacting these desired products. We concluded that we should treat this problem in a more summary fashion, as described in the following paragraphs.

First, we decided to describe the process of individual data entry required during the specified real-time processing in a generic fashion, since each data item input was essentially processed in the same way. That is, regardless of which data item was input, it was to be subjected to a format and legal-value check, and an error message was to be produced if incorrect. If the input was correct, the data was to be retained and the next data input prompt was to be displayed. Or if an error was detected, the current prompt was to be redisplayed. There also was the need to allow the operator to skip a prompt if it described an optional input data item, or to allow him to select the previous prompt if he desired.

In order to describe the generic approach of processing data inputs, it was necessary to accomplish bookkeeping of information concerning what data item is being processed, what the next prompt should be, what the last prompt was, what the legal value and format for the item being input was, what error code was appropriate if an error was detected, plus other information necessary to allow the DP to recognize what process was underway and what process was to follow. This capability was implemented by grouping the needed bookkeeping information into ENTITY_CLASSES, and treating the input of each data item as a single generic input MESSAGE. The result can be seen in the regeneration of the SAMS requirement in Appendix B.

The generic impact approach for data input, as described above, allowed the remainder of our software engineering to treat the processing described in the DFSR in a more summary fashion. Each real-time process (XMA, XMB, etc.) was treated as if the stimulus for its processing was a single MESSAGE (as defined in Annex A of the DFSR) and each such MESSAGE

which was MADE BY DATA for which all values were legal and correctly formatted. This was feasible, since each of the individual data items in the MESSAGE had initially been subjected to the tests we described in the generic process for testing these individual inputs. Thus, the process described in the net for XMA assumes that all the DATA encountered is of the correct format and possesses a legal value. As a result, the processing logic described in the R_NET assumes "correct" data and, therefore, does not include the processing shown in the DLTs to assure that the data are correct and legal, nor the prompts for the next operator entry. As stated, all of this processing was covered in the generic definition of the input process.

There were occasions, however, where the format of an input DATA item was not constant and, therefore, could not be checked during the generic input process because specific processing was necessary to determine the correct format. For example, in the XMA and XMB processing, the PRT_NO_FLD may have any of several formats, depending on whether it contains a National Stock Number, a manufacturer's part number, or a commercial vehicle code. In a case such as this, different format checks are necessary and the processing described in the R_NET must determine which format is appropriate and whether the DATA item actually has that format. All other DATA items in the input MESSAGES that did not possess variable formats were subjected to the generic input processing previously described.

This approach served to reduce the application time for developing the requirements data base so that the desired products of this demonstration could be attained. However, this approach (if applied to a "real" system under development) might create some problems, in that R_Nets so derived don't literally match the approach outlined in the specification. Although the approach used is a true representation of the overall required processing, the literal (rather than generic) representation of each data item entry as a separate MESSAGE would be preferable for verification of an actual system under development. It should be recognized, however, that added resources (including more time) would be necessary for application of SREM at the greater level of detail that would be required.

3.2.3 Impact of the Selected Approaches

Based on the size and scope of the MOM DFSR, it is clear that many man-years were devoted to its completion. Considerable manual effort was clearly needed to organize, cross-reference, and produce the specifications. A very small portion of the probable original effort expended on this specification was available to accomplish this SREM engineering analysis. Although time pressures required us to modify our approach slightly, the resulting requirements data base and its RADX evaluation is complete and we have high confidence that we have identified all the important deficiencies, and nearly all of the less important ones that exist in the MOM DFSR Decision Logic Tables. Thus, our applied approach has provided the products desired under this contract in every particular.

3.3 DEVELOPMENT OF INTERFACE ELEMENTS

The initial phase in defining the requirements of the MOM DFSR required the identification of the INPUT_INTERFACEs and OUTPUT_INTERFACEs which connect the data processor (DP) with external devices (SUBSYSTEMs in RSL). Once the interfaces were identified, the MESSAGES passing through them and the contents of these MESSAGES were defined. These items were recorded in the requirements data base as they were identified. Table 3.1 lists the RSL definition of the element types, the relationships, and the complementary relationships used in the development of interface elements. Figure 3-1 illustrates the inter-relationships of these defined items.

Table 3.1 RSL Definitions Used in the Development of Interface Elements

DEFINITION OF ELEMENTS	
SUBSYSTEM	A PART OF THE SYSTEM WHICH COMMUNICATES WITH THE DATA PROCESSING SUBSYSTEM.
INPUT_INTERFACE	A PORT BETWEEN THE DATA PROCESSING SUBSYSTEM AND ANOTHER SUBSYSTEM THROUGH WHICH DATA IS PASSED TO THE DATA PROCESSING SUBSYSTEM.
OUTPUT_INTERFACE	A PORT BETWEEN THE DATA PROCESSING SUBSYSTEM AND ANOTHER PART OF THE SYSTEM THROUGH WHICH DATA IS PASSED TO THE OTHER SUBSYSTEM.
MESSAGE	AN AGGREGATION OF DATA AND FILES THAT PASS THROUGH AN INTERFACE AS A LOGICAL UNIT.
FILE	AN AGGREGATION OF INSTANCES OF DATA, EACH INSTANCE OF WHICH IS TREATED IN THE SAME MANNER.
DATA	A SINGLE PIECE OF INFORMATION OR SET OF INFORMATION REQUIRED IN THE IMPLEMENTED SOFTWARE.
DEFINITION OF RELATIONSHIPS	
CONNECTS TO (CONNECTED TO)	IDENTIFIES WITH WHICH SUBSYSTEM THE INPUT_INTERFACE OR OUTPUT_INTERFACE COMMUNICATES.
PASSES (PASSED THROUGH)	IDENTIFIES THE MESSAGES WHICH ARE PASSED THROUGH THE INTERFACE.
MAKES (MADE BY)	INDICATES THAT THE DATA OR FILE IS A LOGICAL COMPONENT OF THE MESSAGE.
CONTAINS (CONTAINED IN)	IDENTIFIES THE MEMBERS OF EACH INSTANCE IN A FILE; DATA MAY BE CONTAINED IN ONLY ONE FILE.
INCLUDES (INCLUDED IN)	INDICATES A HIERARCHICAL RELATIONSHIP BETWEEN DATA; THAT IS DATA INCLUDES DATA.

AMERICAN
NATIONAL
STANDARD

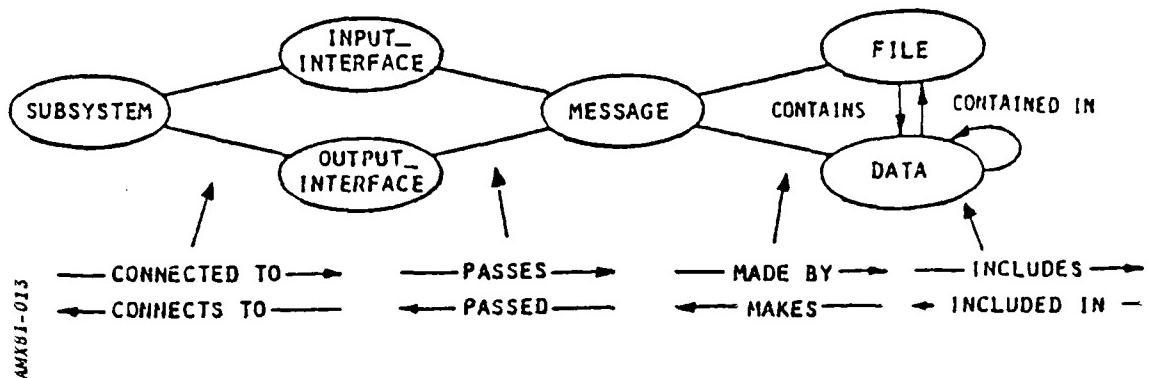


Figure 3-1 Interface Element Interrelationships

3.3.1 Subsystem Definition

To identify the SUBSYSTEMs, we reviewed the text, the functional flow diagrams, and the input and output descriptions. The primary source for identification of the SUBSYSTEMs was the input descriptions of Annex A and the output description of Annex B. The SUBSYSTEMs that have been defined are listed in Table 3.2. These same sources produced the information as to

Table 3.2 SUBSYSTEMs Identified in the MOM_DFSR

<pre> SUBSYSTEM: MOM_CRT. CONNECTED TO: OUTPUT_INTERFACE: TU_MOM_CRT. SUBSYSTEM: MOM_KEYBOARD. CONNECTED TO: INPUT_INTERFACE: FRUM_MOM_KEYBOARD. SUBSYSTEM: MOM_MAG_MEDIA. CONNECTED TO: INPUT_INTERFACE: FROM_MOM_MAG_MEDIA. OUTPUT_INTERFACE: TU_MOM_MAG_MEDIA. SUBSYSTEM: MOM_PRINTER. CONNECTED TO: OUTPUT_INTERFACE: TU_MOM_PRINTER. </pre>

AMX61-067

which SUBSYSTEMs transmitted information to the MOM_USER data processor (INPUT_INTERFACE in RSL) and which received information (OUTPUT_INTERFACE in RSL).

3.3.2 Interface Definition

An INPUT_INTERFACE in RSL denotes a link through which information is communicated into the DP. An OUTPUT_INTERFACE is one through which information is communicated from the DP. The relationship of each type of interface is that it is connected to a SUBSYSTEM. Each interface passes one or more messages, but to prevent ambiguity, the methodology states that a specific MESSAGE may pass only one interface.

3.3.3 MESSAGE Definition

MESSAGES are the aggregation of DATA and FILEs that are communicated as logical units across the interfaces. While these MESSAGES are made by DATA and FILE information, a single DATA item or FILE may be used to make several MESSAGES. The DATA and FILEs that an interface communicates are those that MAKE all the messages that PASS through the interface. A FILE, as used here, is a repetitive set of one or more DATA items. The major source of MESSAGES and their contents also was Annexes A and B.

Each input and output file in Annexes A and B has been defined as an RSL MESSAGE. Where necessary, the RSL MESSAGE name was slightly modified from Input/Output File Names in the Annexes to insure uniqueness. For example, the suffix MSG_IN was added to identify the input messages while the suffix MSG_OUT was utilized in a like manner to identify the output messages. Because RSL allows the use of a synonym to shorten lengthy titles, the unique number provided for each Input/Output description in the DFSR Annexes (e.g., I2 01 KZ) was employed to fulfill this role. Where possible, the data element abbreviations provided in the Annexes were used as the RSL DATA names within each RSL MESSAGE. In order to assure non-ambiguity, each DATA item is named uniquely. It was necessary, therefore, to slightly modify the abbreviated data names in Annexes A and B, and to add the suffixes IN or OUT, respectively.

In RSL, a MESSAGE is MADE BY DATA which may include other DATA. A major summary DATA item made each MESSAGE. It included all of the data elements contained in the input description of Annex A. This allowed use of the summary item in the SREM definition of processing, but the system still understood that all the INCLUDED DATA were involved. An example of all the relationships produced from the requirements data base for the input MESSAGE: WRK_ORD_REGISTRATION_DATA_MSG_IN is shown in Figure 3-2.

MESSAGE: WRK_ORD_REGISTRATION_DATA_MSG_IN
 EQUIVALED TO
 SYNONYM: I2_01_X2
 MADE BY
 DATA: MOM_XTBD_MSG_TYPE
 DATA: WRK_ORD_REGISTRATION_DATA_MSG_IN_INFO
 INCLUDES
 DATA: CUNO_DSG_REIMB_CUST_IN
 DATA: OIC_IN
 DATA: ENO_ITEM_COMP_INO_FLD_IN
 DATA: EQUIP_REDN_CD_IN
 DATA: EQUIP_SER_LCL_CON_NO_IN
 DATA: FILE_INPT_ACT_CD_IN
 DATA: IDENT_NO_CD_IN
 DATA: INTRA_SHOP_CD_IN
 DATA: IPO_IN
 DATA: ITEM_NOMEN_ITEN_NOUN_FLD_IN
 DATA: MAT_REDN_REPT_DSG_IN
 DATA: PRT_NO_FLD_IN
 DATA: SEQ_NO_IN
 DATA: UIC_CUST_IN
 DATA: UIC_SPT_IN
 INCLUDES
 DATA: DESCR_DSG_UIC_IN
 DATA: PRNT_ORG_DSG_UIC_IN
 DATA: SVC_DSG_UIC_IN

ANX81-060

Figure 3-2 Hierarchical Definition of the MESSAGE:
WRK_ORD_REGISTRATION_DATA_MSG_IN

3.3.3.1 Input MESSAGE Definition

Frequently, it was necessary that an Annex A input description be structured into more than one RSL MESSAGE. An example of this situation is provided by the input description for "Work Order Requirements Data". This input description addresses three distinct subjects -- Tasks, Parts, and Supplemental Parts. Each of these subjects requires a separate entry into the DP, and each is MADE BY different DATA items. Thus, under the SREM rules, these are three separate input MESSAGES, and each is subjected to separate processing logic. Accordingly, this requirement with the input title "Work Order Requirements Data" in Annex A resulted in the following three RSL MESSAGES:

```

WRK_ORD_REQMTS_DATA_TASK_MSG_IN
WRK_ORD_REQMTS_DATA_PARTS_MSG_IN
WRK_ORD_REQMTS_DATA_SUPL_PARTS_MSG_IN

```

Table 3.3 provides a comparison of all of the Annex A input descriptions with the resultant RSL INPUT MESSAGES.

Table 3.3 Comparison of Annex A Input Descriptions and Equivalent RSL MESSAGE Names

SAMS DFSR ANNEX A		RSL	
INPUT TITLE	ID CODE	SYNONYM	INPUT MESSAGE NAME
WORK ORDER REGISTRATION DATA	I2_01_KZ	I2_01_KZ	WRK_ORD_REGISTRATION_DATA_MSG_IN
WORK ORDER REGISTRATION ADDITIONAL DATA	I2_02_KZ	I2_02_KZ	WRK_ORD_REGISTRATION_ADDL_DATA_RPR_MSG_IN
		I2_02A_KZ	WRK_ORD_REGISTRATION_ADDL_DATA_CAL_MSG_IN
WORK ORDER REQUIREMENTS DATA	I2_03_KZ	I2_03_KZ	WRK_ORD_REQMTS_DATA_TASK_MSG_IN
		I2_03A_KZ	WRK_ORD_REQMTS_DATA_PARTS_MSG_IN
		I2_03B_KZ	WRK_ORD_REQMTS_DATA_SUPL_PARTS_MSG_IN
WORK ORDER CONSUMPTION DATA	I2_04_KZ	I2_04_KZ	WRK_ORD_CONSUMPTION_DATA_LABOR_MSG_IN
		I2_04A_KZ	WRK_ORD_CONSUMPTION_DATA_PARTS_MSG_IN
		I2_04B_KZ	WRK_ORD_CONSUMPTION_DATA_TASK_MSG_IN
WORK ORDER STATUS DATA	I2_05_KZ	I2_05_KZ	WRK_ORD_STATUS_DATA_MSG_IN
MAINTENANCE PROGRAM DATA	I2_06_KY	I2_06_KZ	MAINT_PROGRAM_DATA_MSG_IN
MAINTENANCE PROGRAM REQUIREMENTS	I2_07_BM	I2_07_BM	MAINT_PROGRAM_REQUIREMENTS_MSG_IN
REPAIR PART MORTALITY DATA	I2_08_BM	I2_08_BM	REPAIR_PART_MORTALITY_DATA_MSG_IN
PART NUMBER CHANGE DATA	I2_12_KY	I2_12_KY	PART_NUMBER_CHANGE_DATA_MSG_IN
PARTS RECEIPTS/STATUS/RECONCILIATION	I2_13_KZ	I2_13_KZ	PRTS_RCPTS_STATUS_RECONCIL_RCPT_MSG_IN
		I2_13A_KZ	PRTS_RCPTS_STATUS_RECONCIL_RESPON_MSG_IN
		I2_13B_KZ	PRTS_RCPTS_STATUS_RECONCIL_STATUS_MSG_IN
SUPPLY STATUS	I2_15_BD	I2_15_BD	SUPPLY_STATUS_MSG_IN
SHIPMENT STATUS	I2_16_BD	I2_16_BD	SHIPMENT_STATUS_MSG_IN
SSL ADJUSTMENT	I2_17_KY	I2_17_KY	SHOP_STOCK_LIST_ADJUSTMENT_A_MSG_IN
		I2_17A_KY	SHOP_STOCK_LIST_ADJUSTMENT_B_MSG_IN
		I2_17B_KY	SHOP_STOCK_LIST_ADJUSTMENT_C_MSG_IN
		I2_17D_KY	BENCH_STOCK_ADJUSTMENT_D_MSG_IN
		I2_17E_KY	BENCH_STOCK_ADJUSTMENT_E_MSG_IN
		I2_17F_KY	BENCH_STOCK_ADJUSTMENT_F_MSG_IN
		I2_17G_KY	BENCH_STOCK_ADJUSTMENT_G_MSG_IN
SUPPLY RECONCILIATION	I2_18_BM	I2_18_BM	SUPPLY_RECONCILIATION_AN_MSG_IN
		I2_18A_BM	SUPPLY_RECONCILIATION_AP_MSG_IN
WORK ORDER PARTS ADJUSTMENT	I2_20_KR	I2_20_KR	WRK_ORD_PARTS_ADJUSTMENT_MSG_IN
EQUIPMENT RECALL NEW ITEM	I2_20A_KY	I2_20A_KY	EQUIP_RECALL_NEW_ITEM_A_MSG_IN
		I2_20B_KY	EQUIP_RECALL_NEW_ITEM_B_MSG_IN
EQUIPMENT RECALL REQUIREMENTS	I2_23_BM	I2_23_BM	EQUIP_RECALL_REQUIREMENTS_MSG_IN
ALT/ERU REQUIREMENTS	I2_24_BY	I2_24_BY	ALT_ERU_REQUIREMENTS_MSG_IN
FLOAT FILE ADJUSTMENT	I2_40_KY	I2_40_KY	FLOAT_FILE_ADJUSTMENT_MSG_IN
USAGE DATA	I2_50_KR	I2_50_KR	USAGE_DATA_MSG_IN
USAGE DEVICE COMPONENT CHANGE	I2_51_KY	I2_51_KY	USAGE_DEVICE_COMPONENT_CHANGE_MSG_IN

Table 3.3 Comparison of Annex A Input Descriptions and Equivalent RSL MESSAGE Names (Continued)

SAMS OFSR ANNEX A		RSL	
INPUT TITLE	ID CODE	SYNONYM	INPUT MESSAGE
USAGE EXCEPTION LIST	I2_52_BR	I2_52_BR	USAGE_EXCEPTION_LIST_MSG_IN
USAGE DATA SURVEY (ANNOTATED)	I2_53_4R	I2_53_4R	USAGE_DATA_SURVEY_ANNOTATED_MSG_IN
TASK PERFORMANCE FACTOR ADJUSTMENT	I2_60_KY	I2_60_KY	TASK_PERFORMANCE_FACTOR_ADJUSTMENT_MSG_IN
WORK CENTER LABOR	I2_70_KY	I2_70_KY	WORK_CENTER_LABOR_MSG_IN
TABLE BUILD	I2_96_KY	I2_96A_KY I2_96B_KY I2_96C_KY I2_96D_KY I2_96E-KY	TABLE_BUILD_ECC_MSG_IN TABLE_BUILD_WRK_REQ_STA_MSG_IN TABLE_BUILD_STOCK_STOCKAGE_LEVEL_MSG_IN TABLE_BUILD_INQUIRY_ACTION_MSG_IN TABLE_BUILD_WORK_CENTER_MSG_IN
INQUIRY	I2_97_KY	I2_97A_KY I2_97B_KY	INQUIRY_MSG_IN INQUIRY_SUMMARY_MSG_IN
PARAMETER	I2_98_KY	I2_98A_KY I2_98B_KY I2_98C_KY I2_98D_KY I2_98E-KY I2_98F_KY I2_98G_KY I2_98H-KY	PARAMETER_FOLLOW_UP_MSG_IN PARAMETER_WORK_ORDER_MSG_IN PARAMETER_WORKLOAD_BACKLOG_AGE_MSG_IN PARAMETER_PARTS_STATUS_DETAIL_MSG_IN PARAMETER_REPORT_CONTROL_MSG_IN PARAMETER_NORS_NORM_DATA_MSG_IN PARAMETER_PREVIOUS_CYCLE_DATE_MSG_IN PARAMETER_DUTY_HOURS_MSG_IN
CROSS REFERENCE TRANSACTION	I2_39_KY	I2_39A_KY I2_99B_KY	CROSS_REFERENCE_TRANSACTION_A_MSG_IN CROSS_REFERENCE_TRANSACTION_B_MSG_IN

Annex 3

3.3.3.2 Output MESSAGE Definition

The output descriptions provided in Annex B were the source for RSL OUTPUT MESSAGES. An example of an output MESSAGE from the requirements data base is shown in Figure 3-3. As with the input descriptions of Annex A, the messages defined by the output descriptions in Annex B sometimes had

AMX01-072

MESSAGE: FLOAT_STATUS_REPORT_MSG_OUT EQUATED TO SYNONYM: 02_10_4Y DOCUMENTED BY SOURCE: SAMS_1_PAGE_33 MADE BY	DATA: AUTMNZD_QNTY_OFN_10_4Y DATA: DATE_PREP_LND_10_4Y DATA: EUH_SET_CH_10_4Y DATA: ITEM_NUMEN_ITEM_NUMN_FLD_10_4Y DATA: OMHAND_QNTY_OFN_10_4Y DATA: PRT_NO_FLD_10_4Y DATA: QTY_ENQR_10_4Y DATA: QTY_EQH_10_4Y DATA: JIC_SPT_10_4Y DATA: UNIT_NAME_SPT_10_4Y
---	---

Figure 3-3 Hierarchical Definition of the Output MESSAGE:
FLOAT_STATUS_REPORT_MSG_OUT

to be divided into more than one RSL MESSAGE. This implied requirement was determined after the completion of an analysis of the hardcopy formats provided in Annex B. These formats show that portions of the outputs are single item entries (such as the heading information), while other portions provide fields of repetitive information. An example of a MESSAGE that does not have to be divided is shown in Figure 3-4. In RSL terms, the heading information is a group of DATA items (UIC_SPT, WORK_CENTER_NR, DATE, etc.). Following the header, there are three groups of repetitive sets of DATA. These groups are: 1) WORK ORDERS IN PROCESS, 2) WORK ORDERS AWAITING WORK CENTER, and 3) WORK ORDER AWAITING PARTS. Within each of these groups, a separate line of information (several DATA items) is printed for each Work Order that meets the criteria for the group. Thus, in RSL, each group is a FILE containing multiple instances of DATA (one instance for each line to be printed under the group). Accordingly, the

(WARTIME ESSENTIAL)

02 02 4D WORK CENTER SUMMARY

Figure 3-4 Format for the Work Center Summary Report

MESSAGE is defined in the requirements data base as shown in Figure 3-5, and accurately depicts the necessary processing to produce it.

MESSAGE: WORK_CENTER_SUMMARY_MSG_OUT
 EQUIATED TO
 SYNONYM: 02_02_40
 MADE BY
 FILE: *WHRK_CEN_SUMM_WRK_AWTG_PARTS_4SG_OUT_INFO
 CONTAINS
 DATA: DATE_ACHT_OHO_OUT_PRT
 DATA: INTRA_SHUP_CD_OUT_PRT
 DATA: IPD_OUT_PRT
 DATA: ITEM_NOMEN_ITEM_NUMN_FLD_OUT_PRT
 DATA: MM_EXP_TEN_OUT_PRT
 DATA: MM_PRJ_TEN_OUT_PRT
 DATA: MM_RMN_TEN_OUT_PRT
 DATA: PHC_WRK_CEN_OUT_PRT
 DATA: SEQ_NO_OUT_PRT
 DATA: TOT_MM_EXP_AWTG_PRT_TEN_OUT_PRT
 DATA: TOT_MM_PRJ_AWTG_PRT_TEN_OUT_PRT
 DATA: TOT_MM_RMN_AWTG_PRT_TEN_OUT_PRT
 DATA: UIC_CUST_OUT_PRT
 DATA: WRK_REQ_STA_CD_OUT_PRT
 DATA: YR_WI_DCD_OUT_PRT
 FILE: *WHRK_CEN_SUMM_WRK_AWTG_SHUP_4SG_OUT_INFO
 CONTAINS
 DATA: DATE_ACHT_OHO_OUT_SHP
 DATA: INTRA_SHUP_CD_OUT_SHP
 DATA: IPD_OUT_SHP
 DATA: ITEM_NOMEN_ITEM_NUMN_FLD_OUT_SHP
 DATA: MM_EXP_TEN_OUT_SHP
 DATA: MM_PRJ_TEN_OUT_SHP
 DATA: MM_RMN_TEN_OUT_SHP
 DATA: PHC_WRK_CEN_OUT_SHP
 DATA: SEQ_NO_OUT_SHP
 DATA: TOT_MM_EXP_AWTG_SHUP_TEN_OUT_SHP
 DATA: TOT_MM_PRJ_AWTG_SHUP_TEN_OUT_SHP
 DATA: TOT_MM_RMN_AWTG_SHUP_TEN_OUT_SHP
 DATA: UIC_CUST_OUT_SHP
 DATA: WRK_REQ_STA_CD_OUT_SHP
 DATA: YR_WI_DCD_OUT_SHP
 FILE: *WHRK_CEN_SUMM_WRK_IN_PROCESS_4SG_OUT_INFO
 CONTAINS
 DATA: DATE_ACHT_OHO_OUT
 DATA: FOL_WRK_CEN_OUT
 DATA: INTRA_SHUP_CD_OUT
 DATA: IPD_OUT
 DATA: ITEM_NOMEN_ITEM_NUMN_FLD_OUT
 DATA: MM_EXP_TEN_OUT
 DATA: MM_OVER_EST_TEN_OUT
 DATA: MM_PRJ_TEN_OUT
 DATA: MM_RMN_TEN_OUT
 DATA: SEQ_NO_OUT
 DATA: TOT_MM_EXP_IN_SHOP_TEN_OUT
 DATA: TOT_MM_OVER_EST_TEN_OUT
 DATA: TOT_MM_PRJ_IN_SHOP_TEN_OUT
 DATA: TOT_MM_RMN_IN_SHOP_TEN_OUT
 DATA: UIC_CUST_OUT
 DATA: YR_WI_DCD_OUT
 MADE BY
 DATA: *WHRK_CEN_SUMM_HEADER_MSG_OUT_INFO
 INCLUDES
 DATA: DATE_WPRH_PRJ_OUT
 DATA: UIC_SPT_OUT
 DATA: UNIT_NAME_SHT_OUT
 DATA: WRK_CEN_CD_OUT
 DATA: *WHRK_CEN_SUMM_TRAILER_4SG_OUT_INFO
 INCLUDES
 DATA: TOT_MM_EXP_WRK_CEN_TEN_OUT
 DATA: TOT_MM_PRJ_WRK_CEN_TEN_OUT
 DATA: TOT_MM_RMN_WRK_CEN_TEN_OUT

ANX81-061

Figure 3-5 Hierarchical Definition of the Output MESSAGE:
WORK_CENTER_SUMMARY_MSG_OUT

A problem arises, however, in properly defining the processing for the output description: Equipment Recall Schedule, as shown in Figure 3-6. In this format, a multiple listing contains several end items of equipment being recalled. Because of its multiple nature, these items would normally be CONTAINED in a FILE. For each such end item, a multiple list of Work Order numbers (one for each item of that type being recalled) exists which includes other data to be printed along with each Work Order Number on multiple lines. This also would normally be organized as a FILE which CONTAINS all the DATA items to be printed on each Work Order Number line. A problem exists because the formal foundations of SREM do not allow a FILE within another FILE, as is suggested by the FILE of work order information lines within each instance of the FILE of end items as shown by the format of this report. To handle this situation unambiguously, this output must be treated in RSL as two MESSAGES. The first provides the one-time provision of header data, which is illustrated in Figure 3-7. The second MESSAGE is MADE BY the two DATA items for the part number and the end item nomenclature, plus a FILE which CONTAINS multiple sets of the DATA for the Work Order Number, the item's serial number, the maintenance code, and the equipment location. The RSL implementation of the second MESSAGE is shown in Figure 3-8. Table 3.4 compares all the Annex B message names to their RSL equivalents.

EQUIPMENT RECALL SCHEDULE

210-13KNA

Figure 3-6 Format for the Equipment Recall Schedule Report

AMX81-062

```
MESSAGE EQUIP_RECALL_SCHEDULE_HEADER_MSG_OUT.  
FORMED BY ALPHA PREP_EQP_RCL_SCH_HEADER_INFO_MSG.  
MADE BY  
  DATA DATE_PREP_ORD_22_4M  
  DATA UNIT_NAME_SPT_22_4M  
  DATA UIC_SPT_22_4M  
  DATA UNIT_NAME_CUST_22_4M  
  DATA UIC_CJST_22_4M .  
EQUATED TO SYNONYM 02_22_4M.
```

Figure 3-7 Hierarchical Definition of the Output MESSAGE:
EQUIP_RECALL_SCHEDULE_HEADER_MSG_OUT

AMX81-063

```
MESSAGE EQUIP_RECALL_SCHEDULE_MSG_OUT.  
FORMED BY ALPHA PREP_EQP_RCL_SCH_22_4M_MSG.  
MADE BY DATA PRT_NO_FLD_22_4M  
  DATA ITEM_NUMEN_ITEM_NOUN_FLD_22_4M  
  FILE EQUIP_RECALL_SCH_OUT.  
EQUATED TO SYNONYM 02_22A_4M.  
FILE EQUIP_RECALL_SCH_OUT.  
CONTAINS  
  DATA EQUIP_SER_LCL_CON_NO_FLD_22_4M  
  DATA RQR_MAINT_CD_22_4M  
  DATA MAINT_SCD_SVC_DATE_ORD_22_4M  
  DATA EQUIP_LOC_22_4M  
  DATA WRK_OCR_NO_22_4M.
```

Figure 3-8 Hierarchical Definition of the Output MESSAGE:
EQUIP_RECALL_SCHEDULE_MSG_OUT

Table 3.4 Comparison of Annex B Output Description and Equivalent RSL
MESSAGE names

SAMS ANNEX 3		ESL	
OUTPUT TITLE	ID CODE	SYNONYM	OUTPUT MESSAGE TITLE
WORK ORDER SUMMARY	32_01_40	32_01A_40 32_01B_40 32_01C_40 32_01D_40	WORK_ORDER_SUMMARY_HEADER_MSG_OUT WORK_ORDER_SUMMARY_IN_SHOP_MSG_OUT WORK_ORDER_SUMMARY_AMT_SHOP_MSG_OUT WORK_ORDER_SUMMARY_AMT_PART_MSG_OUT
WORK CENTER SUMMARY	32_02_40	32_02_40	WORK_CENTER_SUMMARY_MSG_OUT
WORKLOAD SUMMARY BY EQUIPMENT CATEGORY	32_03_40	32_03A_40 32_03B_40 32_03C_40 32_03D_40 32_03E_40	WORKLOAD_SUM_EQUIP_CATEGORY_HEADER_MSG_OUT WORKLOAD_SUM_EQUIP_CAT_PRODUCTION_SUM_MSG_OUT WORKLOAD_SUM_EQUIP_CAT_BACKLOG_AGE_MSG_OUT WORKLOAD_SUM_EQUIP_CAT_BACKLOG_STATUS_TOT_MSG_OUT WORKLOAD_SUM_EQUIP_CAT_EQUIP_STATUS_EQUIP_MSG_OUT
WORK ORDER REGISTER/CLOSED	32_04_4M	32_04_AM_HEAD 32_04_AM	WORK_ORDER_REGISTER_CLOSED_HEAD_MSG_OUT WORK_ORDER_REGISTER_CLOSED_MSG_OUT
WORK ORDER REGISTER/STATUS	32_05_40	32_05_40	WORK_ORDER_REGISTER_STATUS_MSG_OUT
CUSTOMER WORK ORDER RECONCILIATION	32_06_4M	32_06_AM_PART_I 32_06_AM_PART_II	CUSTOMER_WORK_ORDR_DPN_RECONCILIATION_MSG_OUT
WORKS/NORM DATA	32_07_4M	32_07_AM_HEAD 32_07_AM_BODY 32_07_AM_BODY	WORKS_NORM_DATA_HEADER_MSG_OUT WORKS_NORM_DATA_BODY_MSG_OUT WORKS_NORM_DATA_BODY_MSG_OUT
MAINTENANCE PROGRAM CONTROL DOCUMENT	32_08_4M	32_08_AM	MAINT_PROG_CTRL_DOCUMENT_MSG_OUT
MAINTENANCE PROGRAM STATUS REPORT	32_09_4M	32_09_AM	MAINT_PROG_STATUS_REPORT_MSG_OUT
FLOAT STATUS REPORT	32_0_AY	32_0_AM	FLOAT_STATUS_REPORT_MSG_OUT
FLOAT CANDIDATE REPORT	32_1_AY	32_1_AM_HEAD 32_1_AM_BODY	FLOAT_CANDIDATE_REPORT_HEADER_MSG_OUT FLOAT_CANDIDATE_REPORT_BODY_MSG_OUT
WORKLOAD STATUS/AGE	32_2_4M	32_2_AM_OTHER 32_2_AM_AMT 32_2_AM_AMT_VSP 32_2_AM_AMT_SHOP 32_2_AM_AMT 32_2_AM_AMT_VSP_SHOP 32_2_AM_AMT_VSP 32_2_AM_AMT_PARTS	CUSTOMER_WORK_ORDR_OTHER_RECONCILIATION_MSG_OUT CUSTOMER_WORK_ORDR_AMT_MSG_OUT CUSTOMER_WORK_ORDR_FINAL_VSP_RECONCILIATION_MSG_OUT CUSTOMER_WORK_ORDR_AMT_SHOP_RECONCILIATION_MSG_OUT CUSTOMER_WORK_ORDR_AMT_VSP_SHOP_RECONCILIATION_MSG_OUT CUSTOMER_WORK_ORDR_AMT_VSP_RECONCILIATION_MSG_OUT CUSTOMER_WORK_ORDR_AMT_PARTS_RECONCILIATION_MSG_OUT

Table 3.4 Comparison of Annex B Output Description and Equivalent RSL
MESSAGE names (Continued)

SAMS ANNEX B		RSL	
OUTPUT TITLE	ID CODE	SYNONYM	OUTPUT MESSAGE TITLE
ALT/SRD SCHEDULE	32_20_42	32_20_48 HEAD	ALT_SRD_HEAD_MSG_OUT
		32_20_48	ALT_SRD_MSG_OUT
ALT/SRD APPLICATION REPORT	32_21_46	32_21_48 HEAD	ALT_SRD_APPL_HEADER_MSG_OUT
		32_21_48	ALT_SRD_APPLICATION_REPORT_MSG_OUT
EQUIPMENT RECALL SCHEDULE	32_22_46	32_22_48 HEAD	EQUIP_RECALL_SCHEDULE_HEADER_MSG_OUT
		32_22_48	EQUIP_RECALL_SCHEDULE_MSG_OUT
EQUIPMENT RECALL DELINQUENCY LIST	32_23_46	32_23_48 HEAD	EQUIP_RECALL_DELINQUENCY_LIST_HEADER_MSG_OUT
		32_23_48	EQUIP_RECALL_DELINQUENCY_
PART STATUS DETAIL	32_30_46	32_30_48 HEAD	PARTS_STATUS_DETAIL_HEADER_MSG_OUT
		32_30_48	PARTS_STATUS_DETAIL_SUB_HEAD_MSG_OUT
		32_30_48	PARTS_STATUS_DETAIL_JOB_MSG_OUT
		32_30_48	PARTS_STATUS_DETAIL_PART_MSG_OUT
WORK REQUIREMENTS	32_31_40	32_31_40	WORK_REQUIREMENTS_MSG_OUT
PARTS AWAITING DISPOSITION ACTION	32_32_40	32_32_40 II	PARTS_AWTO_DISPPO_ACTION_EXCESS_MSG_OUT
		32_32_40	PARTS_AWTO_DISPPO_ACT_EXCESS_DUR_MSG_OUT
SSL/WORK ORDER ISSUE CANDIDATE LIST	32_33_47	32_33_47	SSL_WORK_ORDER_ISSUE_CANDIDATE_LIST_MSG_OUT
PART NUMBER MISMATCH	32_34_47	32_34_47 HEAD	PART_NBR_MISMATCH_HEADER_MSG_OUT
		32_34_47	PART_NBR_MISMATCH_MSG_OUT
DAILY SUPPLY TRANSACTIONS	32_35_40	32_35_40 I	DAILY_SUPPLY_TRANSACTIONS_PENDSET_MSG_OUT
		32_35_40	SUPPLY_ACTIVITY_POINTS_CANCEL_UPD_MSG_OUT
		32_35_40	DAILY_SUPPLY_TRANSACTIONS_UPD_STA_MSG_OUT
		32_35_40	DAILY_SUPPLY_TRANSACTIONS_IMPRT_STA_MSG_OUT
DOCUMENT REGISTER FOR OPEN SUPPLY TRANSACTIONS	32_36_46	32_36_48 HEAD	PARTS_STATUS_DETAIL_HEAD_MSG_OUT
		32_36_48	PARTS_STATUS_DETAIL_BODY_MSG_OUT
DOCUMENT REGISTER FOR CLOSED SUPPLY TRANSACTIONS	32_37_46	32_37_48 HEAD	DOC_REG_CLOSED_SUP_TRNS_HEAD_MSG_OUT
		32_37_48	DOC_REG_CLOSED_SUP_TRNS_BODY_MSG_OUT

Table 3.4 Comparison of Annex B Output Description and Equivalent RSL
MESSAGE names (Continued)

SANS ANNEX B		RSL	
OUTPUT TITLE	ID CODE	SYNONYM	OUTPUT MESSAGE TITLE
SHOP STOCK ZERO BALANCE REPORT	32_38_4Y	32_38_4Y HEAD	SHOP_STOCK_LIST_ZERO_BALANCE_REPORT_ REPORT_HEADER_MSG_OUT
		32_38_4Y	SHOP_STOCK_LIST_ZERO_BALANCE_REPORT_ MSG_OUT
SHOP STOCK LIST	32_39_4M	32_39_4M HEAD	SHOP_STOCK_LIST_HEADER_MSG_OUT
		32_39_4M	SHOP_STOCK_LIST_MSG_OUT
SHOP STOCK LOCATOR LISTING	32_40_4B	32_40_4B HEAD	SHOP_STOCK_LOCATOR_LISTING_HEADER_MSG_OUT
		32_40_4B	SHOP_STOCK_LOCATOR_LISTING_MSG_OUT
SHOP STOCK CONSTRAINT REPORT	32_41*4Y	32_41_4Y II	SHOP_STOCK_CONSTRAINT_RPT_PRAM_MSG_OUT
		32_41_4Y	SHOP_STOCK_CONSTRAINT_RPT_FUNDS_MSG_OUT
BENCH STOCK LIST	32_42_4Y	32_42_4Y HEAD	BENCH_STOCK_LIST_HEADER_MSG_OUT
		32_42_4Y	BENCH_STOCK_LIST_MSG_OUT
LABOR UTILIZATION SUMMARY	32_50_4B	32_50_4Y 32_50B_4Y 32_50C_4Y 32_50D_4Y 32_50E_4Y	LABOR_UTILIZATION_SUMMARY_HEADER_MSG_OUT LABOR_UTILIZATION_SUMMARY_WHR_AVAIL_MSG_OUT LABOR_UTILIZATION_SUMMARY_LSR_WRCRN_MSG_OUT LABOR_UTILIZATION_SUMMARY_UNIT_WHR_AVAIL_MSG_OUT LABOR_UTILIZATION_SUMMARY_UNIT_LSR_MSG_OUT
WORK CENTER PERSONNEL ROSTER	32_51_4B	32_51_4B	WORK_CENTER_PERSONNEL_ROSTER_MSG_OUT
LABOR UTILIZATION DETAIL	32_52_3W	32_52_3W	LABOR_UTILIZATION_DETAIL_MSG_OUT
USAGE DATA SURVEY	32_50_4R	32_50_4R	USAGE_DATA_SURVEY_MSG_OUT
USAGE EXCEPTION LIST	32_51_3R	32_51_3R	USAGE_EXCEPTION_LIST_MSG_OUT
WORK ORDER DATA	32_30_3W	32_30_3M 32_30_3W II 32_30_3W	WORK_ORDER_DATA_TASK_MSG_OUT WORK_ORDER_DATA_PARTS_MSG_OUT WORK_ORDER_DATA_REGISTRATION_MSG_OUT
TRANSFER DATA	32_32_3W	32_32A_3W 32_32B_3W 32_32C_3W 32_32D_3W 32_32E_3W 32_32F_3W 32_32G_3W 32_32H_3W 32_32I_3W 32_32J_3W	FER_EQUIP_RECALL_EMITEMLMSG_OUT FER_EQUIP_RECALL_NEMITEMLMSG_OUT FER_FLOAT_FILE_ADJUSTMENT_MSG_OUT FER_WRK_CTR_LABOR_MSG_OUT FER_WRT_SUMMER_CHANGE_DATA_MSG_OUT FER_TASK_DPRF_FACTOR_ADJUSTMENT_MSG_OUT FER_USAGE_DEVICE_COMPONENT_CHANGE_MSG_OUT FER_USAGE_DATA_MSG_OUT FER_CROSS_REF_MX_A_CARD_MSG_OUT FER_CROSS_REF_MX_CARD_MSG_OUT

Table 3.4 Comparison of Annex B Output Description and Equivalent RSL
MESSAGE names (Continued)

SAMS ANNEX B			RSL
OUTPUT TITLE	ID CODE	SYNONYM	OUTPUT MESSAGE TITLE
SUPPLY ACTIVITY REQUIREMENTS	32_33_30	32_33_30_A0	SUP_ACT_RQMTS REP_PRT REQ_TURN_IN_MSG_OUT
		32_33_30_AFT	SUP_ACT_RQMTS REP_PRT REQ_TURN_IN_MSG_OUT
RECONCILIATION EXCEPTION REPORT	32_35_4M	32_35_4M_I	RECONCIL_EXCEPT_RPT_STAT_UPDATE_MSG_OUT
		32_35_4M_II	RECONCIL_EXCEPT_RPT_DUE_OUT_NO_DUE_IN_MSG_OUT
		32_35_4M_III	RECONCIL_EXCEPT_RPT_DUE_IN_OR_RECORD_MSG_OUT
SUPPLY ACTIVITY REQUIREMENTS	32_36_40	32_36A_40	SUPPLY_ACTIV_RQMTS_RPR_PARTS_REQUL_TAKIN_MSG_OUT
		32_36B_40	SUPPLY_ACTIV_RQMTS_CANCEL_FOLUP_MSG_OUT
		32_36C_40	SUPPLY_ACTIV_RQMTS_FOLUP_DODRDO_MSG_OUT
INOPERATIVE EQUIPMENT STATUS DATA	32_39_30	32_39A_30	INOP_EQUIP_STATUS_DATA_REGISTRATION_MSG_OUT
		32_39B_30	INOP_EQUIP_STATUS_DATA_PARTS_MSG_OUT
ERROR EXCEPTION REPORT	32_39_4R	32_39_4R	ERROR_EXCEPTION_REPORT_MSG_OUT

3.4 DEVELOPMENT OF GLOBAL DATA

There are two categories of DATA in RSL: LOCAL and GLOBAL. Data which is LOCAL is associated only with the R_NET in which initialized or in which introduced (by an input MESSAGE or the OUTPUT FROM an ALPHA) and is not accessible to any other R_NET. It exists only until processing in that R_NET is complete. For example, if there is a DATA item called X which is LOCAL to an R_NET, each time that the R_NET is traversed, X will have a value, but each time the value may be different, and only the current value of X is available for any given traversal. GLOBAL DATA, on the other hand, is permanent and can be accessed by any R_NET any time it is traversed.

The concept ENTITY is used as a means of organizing the global data in REVS. An ENTITY_CLASS is a repetitive data set which is meant to correspond to some real object or conceptual entity which is of concern to the DP, and an ENTITY_TYPE is a sub-classification within an ENTITY_CLASS. An ENTITY_CLASS is COMPOSED OF ENTITY_TYPES. ENTITY_CLASSES and ENTITY_TYPES have GLOBAL DATA and FILEs ASSOCIATED WITH them. DATA or FILEs which are common to all the ENTITY_TYPES in the ENTITY_CLASS are ASSOCIATED WITH the ENTITY_CLASS, and each ENTITY_TYPE in the ENTITY_CLASS can have different DATA or FILEs ASSOCIATED WITH it. An ENTITY_CLASS or ENTITY_TYPE usually possesses several instances of the set of DATA and FILEs ASSOCIATED with it. The definitions for RSL elements and relationships which apply in developing the GLOBAL data for the MOMs DFSR are given in Table 3.5, and their interrelationships are illustrated in Figure 3-9.

Some GLOBAL DATA or FILEs are not associated with an ENTITY_CLASS or ENTITY_TYPE because there is never more than one instance of them maintained GLOBALLY in memory. These may be GLOBAL flags, GLOBAL lists, GLOBAL constants, or any other DATA items whose values change from time to time, but for which no more than one value is ever resident at one time. In the MOMs DFSR, an example is CURRENT_DATE which is used as the source of information for transfer as DATA being stored in an ENTITY_CLASS or ENTITY_TYPE during certain processing.

The ENTITY_CLASSES and ENTITY_TYPES were developed primarily from Annex D of the MOM DFSR. Each data base description in Annex D was treated as an ENTITY_CLASS with at least one ENTITY_TYPE being assigned to each. SYNONYMs were assigned to each ENTITY_CLASS, using the File ID from the

Table 3.5 RSL Definitions Used in the Development of ENTITY_CLASSES and ENTITY_TYPES

DEFINITION OF ELEMENTS	
ENTITY_CLASS	A GENERAL CATEGORY OF OBJECTS OUTSIDE THE DATA PROCESSING SUBSYSTEM. THE OBJECT ARE THOSE IN THE ENVIRONMENT ABOUT WHICH THE DATA PROCESSING SUBSYSTEM MUST MAINTAIN INFORMATION.
ENTITY_TYPE	A SUBSET WITHIN A GENERAL CLASS (ENTITY_CLASS) OF OBJECTS OUTSIDE THE DATA PROCESSING SUBSYSTEM ABOUT WHICH THE DATA PROCESSOR MUST MAINTAIN INFORMATION.
FILE	AN AGGREGATION OF INSTANCES OF DATA, EACH INSTANCE OF WHICH IS TREATED IN THE SAME MANNER.
DATA	A SINGLE PIECE OF INFORMATION OR SET OF INFORMATION REQUIRED IN THE IMPLEMENTED SOFTWARE.
DEFINITION OF RELATIONSHIPS	
COMPOSES (COMPOSED OF)	IDENTIFIES TO WHICH ENTITY_CLASS AN ENTITY_TYPE BELONGS.
ASSOCIATES (ASSOCIATED WITH)	IDENTIFIES WHICH DATA AND FILES COME INTO EXISTENCE WHEN A DATA PROCESSING STEP (AN ALPHA) EITHER CREATES AN INSTANCE OF AN ENTITY CLASS OR SETS THE ENTITY_TYPE OF AN INSTANCE OF AN ENTITY_CLASS.
CONTAINS (CONTAINED IN)	IDENTIFIES THE MEMBERS OF EACH INSTANCE IN A FILE; DATA MAY BE CONTAINED IN ONLY ONE FILE.
INCLUDES (INCLUDED IN)	INDICATES A HIERARCHICAL RELATIONSHIP BETWEEN DATA; THAT IS DATA INCLUDES DATA.

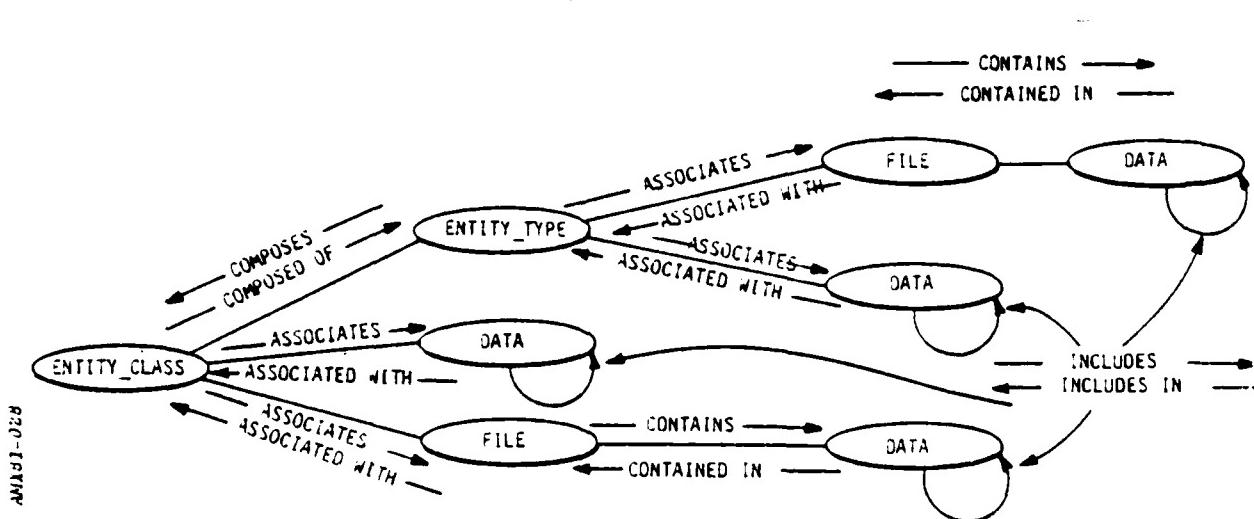


Figure 3-9 GLOBAL DATA and FILE Interrelationships

equivalent file description in Annex D. Table 3.6 provides a comparison of the files in Annex D and RSL equivalent ENTITY_CLASSES. An example of a completely defined ENTITY_CLASS from the requirements data base is shown in Figure 3-10.

An instance of an ENTITY_CLASS is equivalent to a record in a file of Annex D. A new instance of an ENTITY_CLASS (a record) is CREATED BY an ALPHA in an R_NET, and the data stored in the new instance of the ENTITY_CLASS is typically provided by one of the input MESSAGES described as real-time processing (e.g., XMA, XMB, etc.).

A particular instance of an existing ENTITY_CLASS or ENTITY_TYPE may be SELECTED in an R_NET or SUBNET such that a certain boolean condition exists. An example of the use of the SELECT node is shown in Figure 3-11a. Once selected, all the DATA and FILE information ASSOCIATED with the selected ENTITY_CLASS or ENTITY_TYPE remains available until another instance of the same ENTITY_CLASS or ENTITY_TYPE is selected, or the processing on the R_NET on which SELECTed is completed.

A sequence of SELECTs can be defined in an R_NET or SUBNET by using the FOR EACH node. This may be non-conditional, as shown in Figure 3-11b, which means that all instances of the defined repetitive set (an ENTITY_CLASS, an ENTITY_TYPE, or a FILE) are each selected in turn and subjected to the processing described by the ALPHA or SUBNET which follows the FOR EACH node (the ALPHA and SUBNET are the only legal nodes that can follow a FOR EACH). The FOR EACH may also be conditionally constrained, as illustrated in Figure 3-11c. In such a case, only the members of the repetitive set for which the conditional statement is true are selected in turn for the indicated processing of the following ALPHA or SUBNET.

Finally, at some point in the processing, ENTITY_CLASS instances will typically no longer be needed and should be purged. This is accomplished by selecting the appropriate instance of the ENTITY_CLASS which can then be DESTROYED BY an ALPHA.

Table 3.6 Comparison of Annex D Files and Equivalent RSL ENTITY_CLASSES and ENTITY_TYPES

SANS ANNEX D	RSL	ENTITY CLASS	ENTITY_TYPE
FILE TITLE	FILE_ID	SYNONYM	
CROSS REFERENCE FILE	F2_01_BP	F2_01_BP_XREF	CROSS_REFERENCE_FILE
WORK ORDER REGISTRATION	F2_02_BP	F2_02_BP	WORK_ORDER_REGISTRATION
TASK, PART AND REQUISITION	F2_03_BP	F2_03_BP	TASK_PART_REQUISITION
DAILY ACCUMULATED BATCH STORAGE	F2_04_BI_DABs	DAILY_ACCUMULATED_BATCH_STORAGE_DABs	BENCH_STOCK_ADJUSTMENT_XMP_D_DABs BENCH_STOCK_ADJUSTMENT_XMP_E_DABs BENCH_STOCK_ADJUSTMENT_XMP_F_DABs BENCH_STOCK_ADJUSTMENT_XMP_G_DABs CROSS_REFERENCE_TRANSACTIONS_XMP_A_DABs CROSS_REFERENCE_TRANSACTIONS_XMP_B_DABs EQUIPMENT_RECALL_NEW_ITEM_XMP_A_DABs EQUIPMENT_RECALL_NEW_ITEM_XMP_B_DABs FLOAT_FILE_ADJUSTMENT_XMP_F_DABs PARAMETER_DUTY_HOURS_XMP_H_DABs PARAMETER_FOLLOWUP_XMP_AZ_DABs PARAMETER_MONTHLY_DATA_XMP_F_DABs PARAMETER_PARTS_STATUS_DETAIL_XMP_D_DABs PARAMETER_PREVIOUS_CYCLE_DATE_XMP_G_DABs PARAMETER_REPORT_CONTROL_XMP_E_DABs PARAMETER_WORKLOAD_BACKLOG_AGE_XMP_C_DABs

ANNEX-D-05

Table 3.6 Comparison of Annex D Files and Equivalent RSL ENTITY_CLASSES
and ENTITY_TYPES (Continued)

SAMS ANNEX D	RSL	ENTITY_TYPE	
FILE TITLE	FILE ID	SYNONYM	ENTITY_CLASS
			PARTNER_WORK_ORDER_AGE_XM2_B_DABS
			PARTS_RECEIPTS_STATUS_RECONCILIATION_XMK_A_DABS
			PART_NUMBER_CHANGE_DATA_XMN_DABS
			SHIPMENT_STATUS_DABS
			SHOP_LIST_ADJUSTMENT_XMP_A_DABS
			SHOP_LIST_ADJUSTMENT_XMP_B_DABS
			SHOP_LIST_ADJUSTMENT_XMP_C_DABS
			SUPPLY_STATUS_DABS
			TASK_PERFORMANCE_FACTOR_ADJUSTMENT_XMI_DABS
			USAGE_DATA_XMU_DABS
			USAGE_DEVICE_COMPONENT_CHANGE_XMN_DABS
			WORK_CENTER_LABOR_XML_DABS
			WORK_ORDER_CONSUMPTION_XMD_DABS
FLOAT	F2_05_8P	F2_05_8P_FLOAT	FLOAT_FILE_FT
SHOP STOCK LIST	F2_06_8P	F2_06_8P_SSL	SHOP_STOCK_LIST_FT
BENCH STOCK LIST	F2_07_8P	F2_07_8P_BSL	BENCH_STOCK_LIST
TRANSFER FILE	F2_08_85	F2_08_85_FF	TRANSFER_FILE
			IF CROSS_REFERENCE_TRANSACTION_XMK_A
			IF CROSS_REFERENCE_TRANSACTION_XMK_B
			IF_EQUIP RECALL_NEW_ITEM_XME_A
			IF_EQUIPMENT RECALL_NEW_ITEM_XME_B
			IF_FLOAT FILE ADJUSTMENT_XMF
			IF_PART NUMBER CHANGE DATA_XMN
			IF_TASK PERFORMANCE_FACTOR_ADJUSTMENT_XMI

ANNEXD-C62

Table 3.6 Comparison of Annex D Files and Equivalent RSL ENTITY_CLASSES and ENTITY_TYPES (Continued)

SAMS ANNEX D	FILE TITLE	FILE ID	SYNOPSIS	ENTITY CLASS	RSL	ENTITY TYPE
						TF_USAGE_DATA_XMV
						TF_USAGE_COMPONENT_CHANGE_XMV
						TF_WORK_CENTER_LABOR_XML
LABOR UTILIZATION DETAIL		F2 10 81	F2 10 81 LUO	LABOR_UTILIZATION_DETAIL		LABOR_UTILIZATION_DETAIL_FT
MASTER PERSONNEL LABOR		F2 11 8P	F2 11 8P MPL	MASTER_PERSONNEL_LABOR		MASTER_PERSONNEL_LABOR_FT
EQUIPMENT RECALL REQUIREMENTS		F2 20 8W	F2 20 8W ERE	EQUIPMENT_RECALL_REQUIREMENTS		EQUIPMENT_RECALL_REQUIREMENTS_FT
ALT/SRO REQUIREMENTS		F2 21 8W	F2 21 8W ALT SRO	ALT_SRO_REQUIREMENTS		ALT_SRO_REQUIREMENTS_FT
Maintenance PROGRAM REQUIREMENTS		F2 22 8W	F2 22 8W	Maintenance_PROGRAM_REQUIREMENTS		Maintenance_PROGRAM_REQUIREMENTS_FT
REPAIR PART MORTALITY		F2 23 8W	F2 23 8W RPM	REPAIR_PART_MORTALITY		REPAIR_PART_MORTALITY_DATA_FT
USAGE EXCEPTION LIST		F2 24 8W	F2 24 8W ULI	USAGE_EXCEPTION_LIST		USAGE_EXCEPTION_LIST_DATA_BASE_FT
LOOK UP TABLES		F2 30 8P	F2 30 8P	LOOK_UP_TABLE		EQUIPMENT_CATEGORY
						INQUIRY_ACTION
						STORAGE_LEVEL
						WORK_CENTER
						WORK_REQUEST_STATUS

ANDB2-052

ENTITY_CLASS: CROSS_REFERENCE_FILE
 EQUATED TO
 SYNONYM: F2_01_BP_XREF
 DOCUMENTED BY
 SOURCE: APP_D_PAGE_D2
 COMPOSED OF
 ENTITY_TYPE: MANEUVER_CUSTOMER_B_CARD
 ASSOCIATES
 DATA: MNVR_CUST_CR_REF_INFO
 INCLUDES
 DATA: AAC_CUST_CRF_B
 DATA: ACCT_PROC_FLD_CUST_CRF_B
 DATA: CARD_DSG_CD_SAMS_CRF_B
 DATA: CONO_DSG_MSTR_REC_CRF_B
 DATA: CONO_DSG_REIMB_CUST_CRF_B
 DATA: UIC_CUST_CRF_B
 DATA: UIC_PRNT_UNIT_CUST_CRF_B
 DATA: UNIT_NAME_CUST_CRF_B
 DATA: UNIT_NAME_PRNT_CUST_CRF_B
 ENTITY_TYPE: SUPPORT_UNIT_A_CARD
 ASSOCIATES
 DATA: SPT_UNIT_CR_REF_INFO
 INCLUDES
 DATA: AAC_SPT_CRF_A
 DATA: ACCT_PROC_FLD_SPT_CRF_A
 DATA: CARD_DSG_CD_SAMS_CRF_A
 DATA: CONO_DSG_REIMB_CUST_CRF_A
 DATA: UIC_PRNT_UNIT_SPT_CRF_A
 DATA: UIC_SPT_INDIC_CRF_A
 DATA: UNIT_NAME_PRNT_SPT_CRF_A
 DATA: UNIT_NAME_SPT_CRF_A
 ASSOCIATES
 DATA: HOR_DATA_ELEMENTS_CR_REF_INFO
 INCLUDES
 DATA: CONO_DSG_REPT_RQMT_CRF
 DATA: FILE_IDENT_NO_CD_CRF
 DATA: INQ_ACT_CD_CRF
 DATA: NORM_WRK_PO_TEV_CRF
 DATA: PCY_CRF
 DATA: PREV_DAY_CYC_DATE_CRF
 DATA: PREV_MO_CYC_DATE_CRF
 DATA: PREV_WKEY_CYC_DATE_CRF
 DATA: REP_END_DATE_ORD_CRF
 DATA: REP_START_DATE_ORD_CRF
 DATA: JIC_SPT_CRF
 INCLUDES
 DATA: DESCRIPTIVE_DESIG_CRF
 DATA: PRNT_ORG_DESIG_CRF
 DATA: SVC_DESIG_CRF

AMX81-064

Figure 3-10 Hierarchical Definition of the ENTITY_CLASS:
CROSS_REFERENCE_FILE

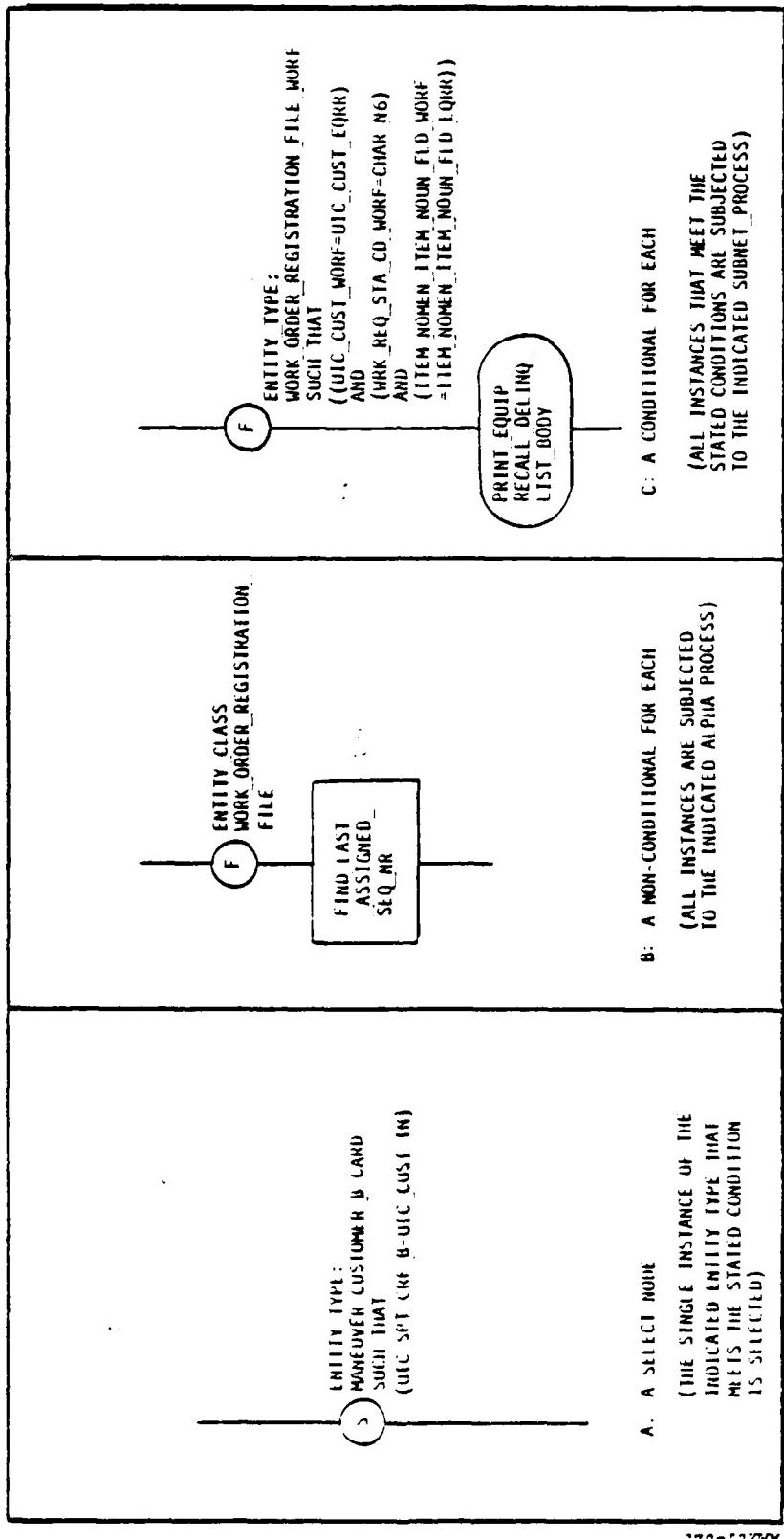


Figure 3-11 Examples of a SELECT and FOR EACH Node

3.5 DEVELOPMENT OF REQUIREMENTS NETS (R_NET)

Once the interfaces, and MESSAGES that cross these interfaces, were identified, analysis of the processing stimulated by the receipt of the input MESSAGES was initiated and documented by development of R_NET and SUBNET structures. Definition of the elements and relationships used in development of R_NETs is shown in Table 3.7. The inter-relationships of these items is provided in Figure 3-12.

3.5.1 R_NET Considerations

Functionally, the R_NETs describe the processing steps (ALPHAs) which must be performed as a result of the arrival of each input MESSAGE, the sequence of the ALPHAs, the logical process branching, and the output MESSAGES which the data processor is required to produce.

The R_NET resembles a traditional flow chart in appearance and represents the processing required in response to each possible stimulus. This stimulus may be either of two types -- the arrival of a MESSAGE from another SUBSYSTEM at the INPUT_INTERFACE on the R_NET, or the occurrence of some EVENT. In the former case, the R_NET is said to be ENABLED by the INPUT_INTERFACE and this interface appears as the first node on the R_NET structure. In the latter case, the R_NET is ENABLED by the EVENT, which itself appears as a node on some R_NET or SUBNET (or on more than one such structure) in the system. The interpretation is that the subject R_NET is ENABLED whenever control reaches the EVENT node on the R_NET or SUBNET on which the EVENT appears.

An ALPHA is a processing step which possesses a single entry and exit path. The ALPHA processes DATA and FILE information which is INPUT TO it, accomplishes the appropriate DATA transformations, and OUTPUTS the resulting DATA and FILE information for GLOBAL storage, for use in subsequent branching logic, for use in following ALPHAs or for producing an output MESSAGE. An ALPHA may also describe other processing. For example, an ALPHA can:

- FORM an output MESSAGE.
- CREATE a new instance of an ENTITY_CLASS.

Table 3.7 RSL Definitions Used in the Development of R_NETS

DEFINITION OF ELEMENTS	
R_NET	A STRUCTURED GRAPH OF LOGICAL PROCESSING STEPS THAT MUST BE PERFORMED BY THE DATA PROCESSING SUBSYSTEM IN RESPONSE TO EXTERNAL OR INTERNAL STIMULI. THE PROCESSING STEPS ARE ALPHAS OR SUBNETS WHICH MAY BE EXPANDED TO LOWER LEVELS OF DETAIL.
SUBNET	A SUBSTRUCTURE OF LOGICAL PROCESSING STEPS THAT MUST BE PERFORMED TO ACCOMPLISH THE REQUIREMENTS OF THE NEXT HIGHER NETWORK (SUBNET OR R_NET) ON WHICH THE SUBNET IS REFERENCED.
ALPHA	A BASIC PROCESSING STEP IN THE FUNCTIONAL REQUIREMENTS.
ENTITY_CLASS	A GENERAL CATEGORY OF OBJECTS OUTSIDE THE DATA PROCESSING SUBSYSTEM. THE OBJECTS ARE THOSE IN THE ENVIRONMENT ABOUT WHICH THE DATA PROCESSING SUBSYSTEM MUST MAINTAIN INFORMATION.
ENTITY_TYPE	A SUBSET WITHIN A GENERAL CLASS (ENTITY_CLASS) OF OBJECTS OUTSIDE THE DATA PROCESSING SUBSYSTEM ABOUT WHICH THE DATA PROCESSOR MUST MAINTAIN INFORMATION.
MESSAGE	AN AGGREGATION OF DATA AND FILES THAT PASS THROUGH AN INTERFACE AS A LOGICAL UNIT.
FILE	AN AGGREGATION OF INSTANCES OF DATA, EACH INSTANCE OF WHICH IS TREATED IN THE SAME MANNER.
DATA	A SINGLE PIECE OF INFORMATION OR SET OF INFORMATION REQUIRED IN THE IMPLEMENTED SOFTWARE.
DEFINITION OF RELATIONSHIPS	
CREATES (CREATED BY)	INDICATES THAT THE ALPHA CREATES AN INSTANCE OF THE ENTITY_CLASS.
DESTROYS (DESTROYED BY)	INDICATES THAT THE ALPHA DESTROYS THE CURRENTLY SELECTED INSTANCE OF THE ENTITY_CLASS.
FORMS (FORMED BY)	INDICATES THAT THE ALPHA ESTABLISHES THE MESSAGE AS THE ONE TO BE PASSED BY THE CORRESPONDING OUTPUT_INTERFACE WHEN THAT INTERFACE IS ENCOUNTERED ON THE NET.
INPUTS (INPUT TO)	IDENTIFIES THE DATA AND FILES USED BY THE ALPHA.
OUTPUTS (OUTPUT FROM)	IDENTIFIES THE DATA AND FILES WHOSE VALUES OR CONTENTS ARE MODIFIED BY THE ALPHA.
SETS (SET BY)	INDICATES THAT THE ALPHA ESTABLISHES THE CURRENTLY SELECTED INSTANCE OF AN ENTITY_CLASS TO BE OF THE ENTITY_TYPE.
CONTAINS (CONTAINED IN)	IDENTIFIES THE MEMBERS OF EACH INSTANCE IN A FILE. DATA MAY BE CONTAINED IN ONLY ONE FILE.
INCLUDES (INCLUDED IN)	INDICATES A HIERARCHICAL RELATIONSHIP BETWEEN DATA, THAT IS DATA INCLUDES DATA.
REFERS REFERRED BY	INDICATES ELEMENT TYPES THAT ARE IN THE STRUCTURE OF A R_NET OR SUBNET.

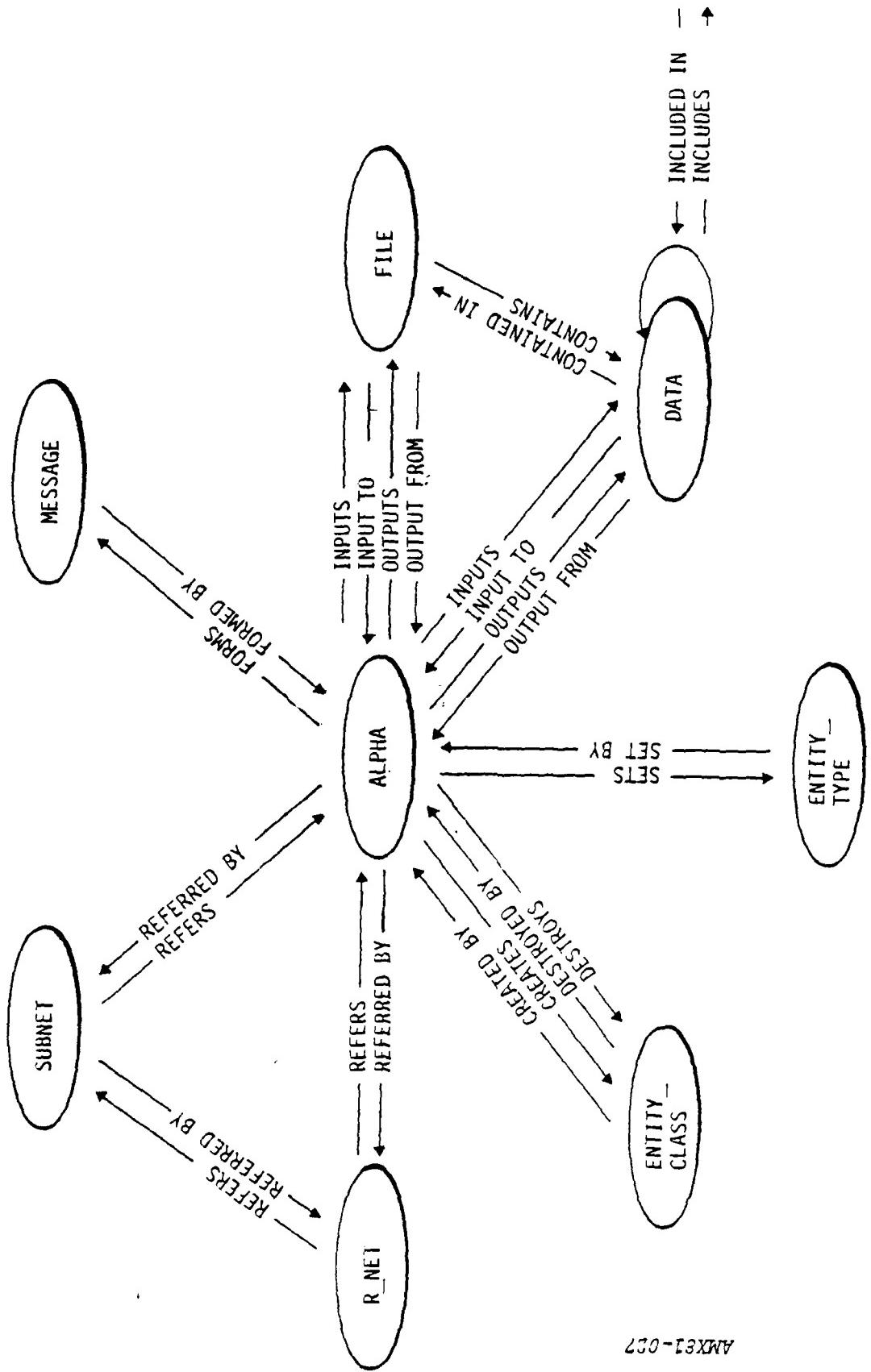


Figure 3-12 R_Net Interrelationships

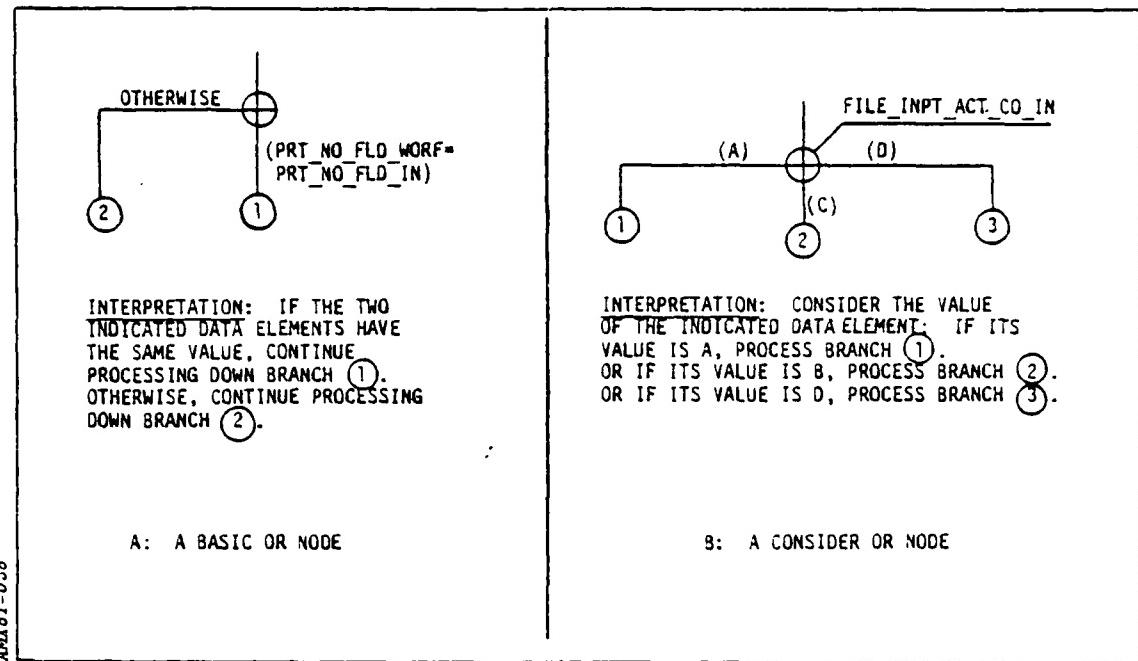
- SET a transformation of an ENTITY_TYPE from one type to another.
- DESTROY an instance of an ENTITY_CLASS (to include its COMPOSED ENTITY_TYPE).

As subsequent analysis of processing continues, it may be determined that some logical branching is appropriate within the ALPHA such that more than one output path could occur. In such a case, it should be replaced by a SUBNET which can contain processing logic. The SUBNET is treated as though the flow path(s) in the SUBNET were physically inserted into the higher level flow path of the structure on which it is located.

More complex flow situations are expressible in RSL by the use of structured nodes which fan-in and fan-out to specify different processing paths. The structured nodes are the AND and OR nodes.

The meaning of an AND structure is that the paths are mutually order-independent. The processes or parallel paths may be executed in any order, or even in parallel. The fan-in at the end of the AND structure is a synchronization point. All of the parallel paths must be completed before any of the processes following the rejoin are performed. There are two types of OR nodes: the basic OR node and the CONSIDER OR node. For the basic OR node, the condition on each exit of the node is a standard Boolean expression which may involve DATA elements and constants. This condition is evaluated when the OR node is reached. The first condition to be evaluated as TRUE specifies the exit branch to be followed. To prevent problems if all specified conditions are FALSE, an OTHERWISE declaration is used to specify a branch to be followed in such a case. Thus, a basic OR node must have one branch with an OTHERWISE declaration. The basic OR node is illustrated in Figure 13a.

The CONSIDER OR node allows branching on the value of DATA which is of TYPE ENUMERATION, or branching on the ENTITY_TYPE of the currently SELECTed ENTITY_CLASS. Each branch of the CONSIDER OR has an associated criterion. DATA with TYPE ENUMERATION have values which are expressed as words. The criteria specify which branch is to be taken, based on the value of the enumerated DATA item under consideration. All of the legal value names must appear once and only once in the branching criteria. Since the criteria for a CONSIDER OR must be exhaustive, an OTHERWISE



AM201-058

Figure 3-13 The Two Kinds of OR Nodes

branch is not allowed. An example of a CONSIDER_OR is shown in Figure 3-13b.

Figure 3-14 shows the R_NET, SUBNET symbology used for SREM structure development. All but the final symbol, the VALIDATION_POINT, were used in the MOM DFSR effort. The VALIDATION_POINT would be used in Phase 6 of SREM which was beyond the scope of this effort.

3.5.2 Example of an R NET Development

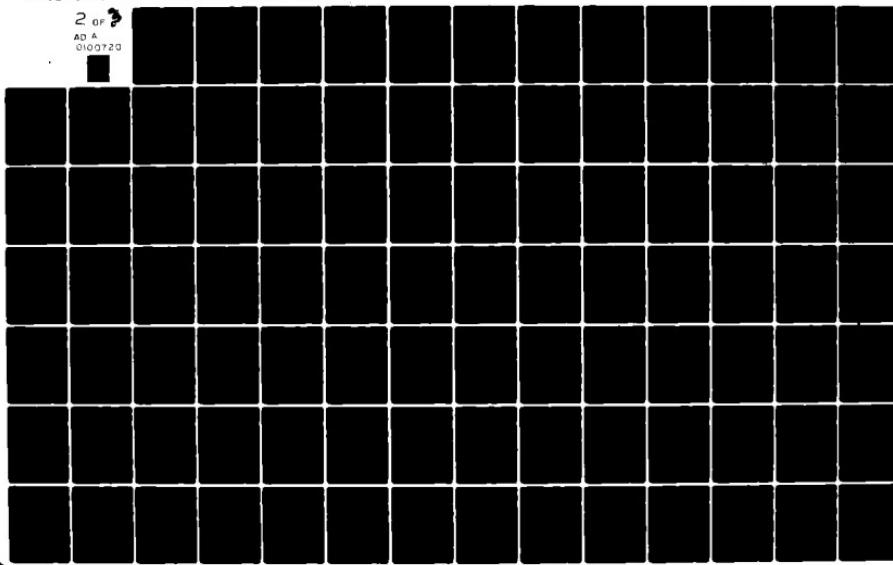
R_NETs are developed in a top down fashion, and any SUBNET defined on the R_NET (or, for that matter, on a SUBNET) is, itself, subjected to definition of the logic of processing it represents. For example, the R_NET: PROCESS_MOM_KEYBOARD_INPUT, Figure 3-15 defines the processing for all input MESSAGES passing the INPUT_INTERFACE: FROM_MOM_KEYBOARD. Note that the first branch decision is accomplished by determining the type of MESSAGE received through the INPUT_INTERFACE. The processing of each type

AD-A100 720

TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2
APPLICABILITY OF SREM TO THE VERIFICATION OF MANAGEMENT INFORMATION—ETC(U)
APR 81 R P LOSHBOUGH, M W ALFORD, J T LAWSON DAHC26-80-C-0020
TRW-37554-6950-001-VOL-1 NL

UNCLASSIFIED

2 OF 2
AD A
0100720



SYMBOLS

ALPHA	-	
AND	-	
ENTRY NODE ON R_NET	-	
ENTRY NODE ON SUBNET	-	
EVENT	-	
FOR EACH	-	
INPUT_INTERFACE	-	
IF OR	-	
CONSIDER OR	-	
OUTPUT_INTERFACE	-	
SELECT	-	
SUBNET	-	
RETURN	-	
TERMINATE	-	
VALIDATION_POINT	-	

AMX81-007

Figure 3-14 R_NET, SUBNET Symbology

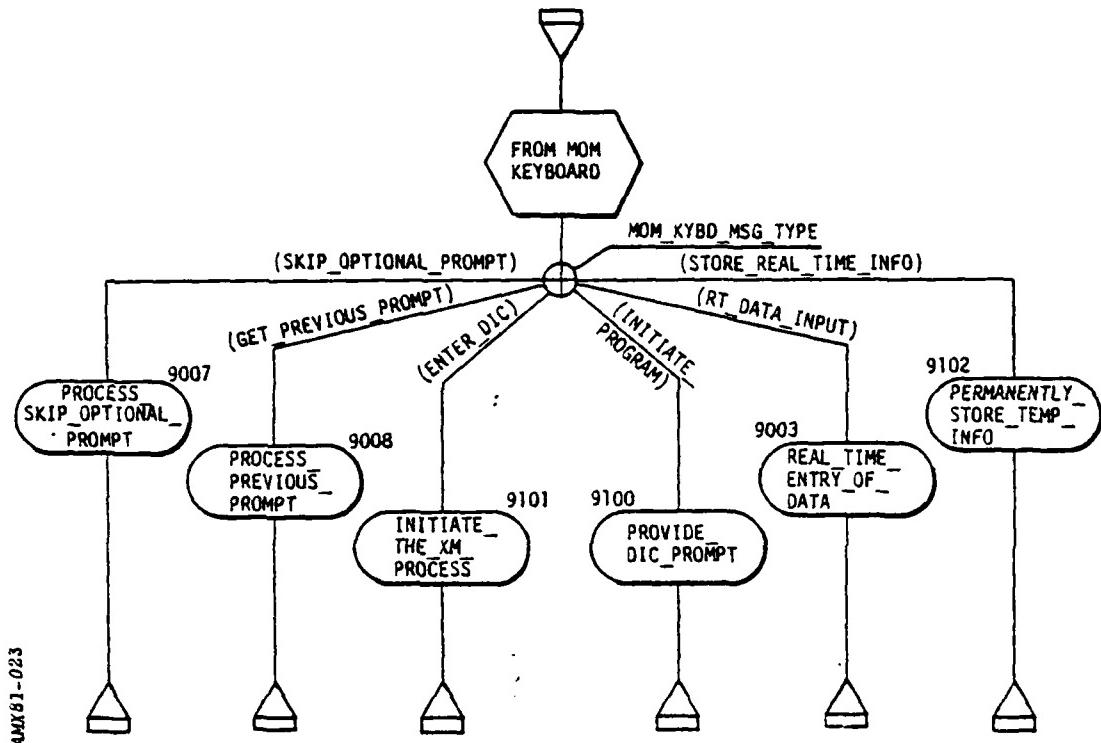


Figure 3-15 RNET: PROCESS_MOM_KEYBOARD_INPUT (NET RT1000)

MESSAGE is defined on a separate processing path by a SUBNET which contains the processing intended on that branch. The decision on which processing path to be taken is based on the value of DATA: MOM_KYBD_MSG_TYPE which is of TYPE ENUMERATION. Thus, the OR node is a CONSIDER OR.

3.5.3 Definition of SUBNET Processing

Each SUBNET is further defined separately. The SUBNET: PERFORMANCE_STORE_TEMP_INFO (not shown) is a complex SUBNET which branches to individual processing paths, depending on the value of the DATA item: DIC (e.g., XMA, XMB, etc.). For example, one branch for XMA contains the SUBNET:

PROCESS_XMA_ENTRY. This SUBNET is shown in Figure 3-16 and is a summary net, in that it is defined as three other SUBNETs. Note that two of the SUBNETs have SYNONYMS shown (e.g., A1001, and A1008). The third SUBNET is indicated as undefined. This is because no processing logic was described for the legal condition where the value of the DATA item FILE_INPT_ACT_CD_IN was D. This is an example of incomplete logic documented in a Trouble Report as a deficiency.

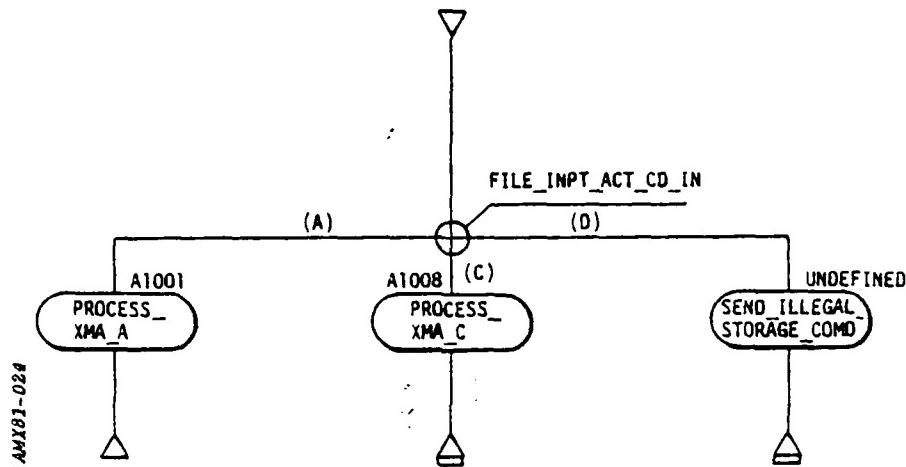


Figure 3-16 SUBNET: PROCESS_XMA_ENTRY (NET A1000)

We can illustrate one level lower by reviewing the definition of SUBNET: PROCESS_XMA_A. This SUBNET which is shown in Figure 3-17, provides the first look at processing at the ALPHA level. In this process, the first node (at A) indicates that a particular instance of GLOBAL information is selected from the ENTITY_TYPE: SUPPORT_UNIT_A_CARD for use in the processing that is to follow. At B a decision is made as to what processing should occur if the selected instance is not found (this is an important consideration which defines a possible error path that typically is not covered in most software specifications ... and wasn't in the MOM DFSR). The term "(FOUND)" is the name of a DATA item that is of TYPE BOOLEAN and which has a VALUE of TRUE or FALSE term. At this point, the "OTHERWISE" indicates the branch to be taken if the VALUE of FOUND is

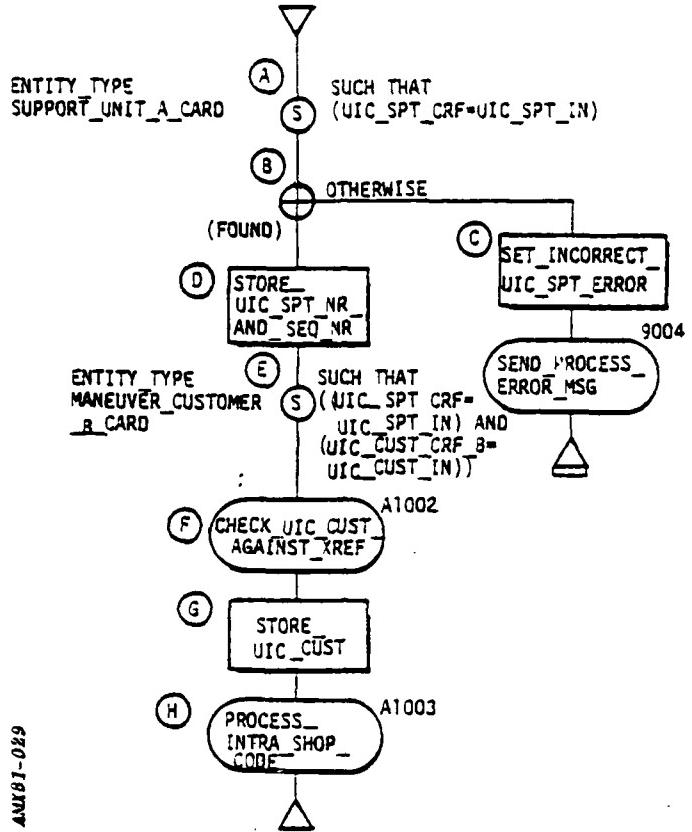


Figure 3-17 SUBNET: PROCESS_XMA_A (NET A1001)

FALSE. In this case, the processing (at C) prepares to produce an output error MESSAGE (which is actually produced in the SUBNET: SEND_PROCESS_ERROR_MSG. Processing is then terminated on this branch for the current stimulus.

If the selected ENTITY_CLASS is found, the UIC_SPT and SEQ_NO values found in the input MESSAGE are stored (D) as GLOBAL DATA in a new instance of the ENTITY_CLASS: WORK_ORDER_REGISTRATION_FILE_WORF. Therefore, the ALPHA: STORE_UIC_SPT_NR_AND_SEQ_NR also CREATES this new instance of the ENTITY_CLASS. Next, (at E) the instance of the ENTITY_TYPE: MANEUVER_CUSTOMER_B_CARD is selected which possesses the same UIC_SPT and UIC_CUST code as those in the input MESSAGE. In this case, the decision as to whether or not the information was found was accomplished at F inside

the SUBNET: CHECK_UIC AGAINST_XREF. The processing in this SUBNET (not shown) indicates that if the information is not found, an informative message is transmitted; otherwise none is transmitted. In either case, UIC_CUST_IN is stored (G) as UIC_CUST_WORF in the same instance of the ENTITY_CLASS: WORK_ORDER-REGISTRATION_FILE_WORF that was CREATED in the preceding ALPHA.

Following this step, the final process is provided by SUBNET: PROCESS_INTRA_SHOP_CD, which assigns the appropriate letter value for DATA: INTRA_SHOP_CD. With the completion of this SUBNET, processing is complete for the stimulus that initiated processing. Thus, the R_NET unambiguously defines all the possible paths of processing that could occur as a result of the receipt of a particular input MESSAGE, in this case, XMA input information where FILE_INPT_ACT_CD_IN had a value of A.

3.5.4 Reason for Liberal Use of SUBNETs

The reader may be wondering about the considerable use of SUBNETs in the definition of processing. The reason for this, and the description of the two basic uses of SUBNETs are provided in the next two paragraphs.

Often, similar processing is required in several different R_Nets or SUBNETs. In such a case, a SUBNET is defined. The SUBNET processing logic is defined just once, and then it is REFERRED (used) as a node on any structure where that processing is needed.

Its second use is for conservation of space in defining structures which contain considerable processing logic. Such nets could be fully defined to the lowest level of detail on a single page, but if they were, the resulting net would be very complex and may be more difficult to understand than the more summary description available by using SUBNETs.

Perhaps of more importance, there is an adverse CALCOMP plot impact for large nets because of the limit of CALCOMP-plot paper size. Because the entire net will be drawn within the boundaries of the available paper size, structures with large quantities of nodes will be drawn with the nodes so small that the names of ALPHAs and SUBNETs (which are also included on the CALCOMP plot) will be unreadable. Appropriate use of SUBNETs to summarize portions of the processing logic alleviates this CALCOMP problem.

3.5.5 Development of the ALPHA Description Sheet

Software engineers accomplish a parallel activity along with developing the R_NETs, and that's the documentation of the DATA flow through the R_NET on a worksheet called the ALPHA Description Sheet. Figure 3-18 shows this worksheet for the SUBNET: PROCESS_XMA_A, whose processing was just described. The ALPHA Description Sheet defines DATA and FILE information INPUT TO and OUTPUT FROM each ALPHA on the R_NET or SUBNET covered by the worksheet. Consistent use of DATA names are used in accordance with their input source and output destination. That is, if the source of a DATA item was the input MESSAGE, its name must be the same as that which MAKES that MESSAGE. On the illustrated worksheet, the three different described ALPHAs are the same ones defined on SUBNET: PROCESS_XMA_A found in Figure 3-17.

The DATA transformations can be observed on the worksheet. For example, the first listed ALPHA inputs two DATA items and outputs three. The order in which the DATA is listed defines the intended transformation. For example, the value for the input DATA item UIC_SPT_IN from the input MESSAGE, having the SYNONYM I2_01_KZ (abbreviated on the worksheet as 01_KZ), is stored in the two DATA items UIC_SPT_WON_WORF (a portion of the work order number) and UIC_SPT_WORF. Both of these DATA items are stored as GLOBAL information in the ENTITY_CLASS: WORK_ORDER_REGISTRATION_FILE_WORF, which has the SYNONYM F2_02_8P (abbreviated here as 02).

Because no entry is shown in the column "Value or Enumeration", whatever value is INPUT by UIC_SPT_IN will now reside in UIC_SPT_WON_WORF, and in UIC_SPT_WORF. Note, however, that in the second ALPHA, only one DATA item (ERROR_CODE) is shown, and it is OUTPUT by the ALPHA. Since there is no input DATA item with a value to be transferred, it is necessary to indicate the value that is to be assigned by this ALPHA to ERROR_CODE. This value is WRONG_UIC_SPT and is used in the SUBNET: SEND_PROCESS_ERROR_MSG (which follows this ALPHA on the SUBNET in Figure 3-17) where a determination is made as to what error text will be output to the operator. This item is local because it will be used before processing in this SUBNET is complete, and will not be needed thereafter.

This ALPHA Description Sheet also documents two other important considerations, as shown at the right side of the worksheet for the first

380-28XWV

Figure 3-18 ALPHA Description Sheet for the SUBNET: PROCESS_XMA_A

ALPHA. These two entries indicate that the ALPHA: STORE_UIC_SPT_NR_AND_SEQ_NR, in addition to the input and output of DATA, also CREATES a new instance of the ENTITY_CLASS: WORK_ORDER_REGISTRATION_FILE_WORF and SETS the new instance to ENTITY_TYPE: WORK-ORDER-REGISTRATION_FILE_CURR (the current work order file).

3.5.6 Entry of R NETs and SUBNETs Into the Requirements Data Base

Upon completion of the manual R_NET development and the ALPHA Description sheets, these efforts were described in RSL and entered into the requirements data base. The two-dimensional structures were defined in a one-dimensional stream of RSL. For example, the SUBNET illustrated in Figure 3-17 was defined in RSL, as shown in Figure 3-19. This effort was

ANX81-163

```
LIST PROCESS_XMA_A.  
-----  
  
SUBNET: PROCESS_XMA_A.  
STRUCTURE:  
  SELECT ENTITY_TYPE: SUPPORT_UNIT_A_CARD SUCH THAT  
    (UIC_SPT_CRF=UIC_SPT_IN)  
    IF (FOUND)  
      ALPHA: STORE_UIC_SPT_VR_AND_SEQ_NR  
      SELECT ENTITY_TYPE: MANEUVER_CUSTOMER_B_CARD SUCH THAT  
        (UIC_SPT_CRF=UIC_SPT_IN) AND  
        (UIC_CUST_CRF=UIC_CUST_IN)  
      SUBNET: CHECK_UIC_CUST_AGA(YST_XREF)  
      ALPHA: STORE_UIC_CUST  
      SUBNET: PROCESS_INTRA_SHOP_CODE  
      RETURN  
    OTHERWISE  
      ALPHA: SET_INCORRECT_UIC_SPT_ERROR  
      SUBNET: SEND_PROCESS_ERROR_MSG  
      TERMINATE  
    END  
  ENDO.
```

Figure 3-19 RSL Listing of the Structure for SUBNET: PROCESS_XMA_A

followed by an RSL definition of the information on the ALPHA Description Sheet, and the entry of this information into the requirements data base. The result of these RSL entries for the three ALPHAs in the SUBNET in Figure 3-17 is displayed in Figure 3-20. Note that two of the ALPHAs are used (REFERRED) on other SUBNETs beside PROCESS_XMA_A. This illustrates

(RAOX COMMANDS
 LIST XMA_A_ALPHAS.

 ALPHA: SET_INCORRECT_UIC_SPT_ERROR.
 OUTPUTS:
 DATA: ERROR_CODE.
 REFERRED BY:
 SUBNET: PROCESS_XMA_A
 SUBNET: PROCESS_XMA_C.
 ALPHA: STORE_UIC_CUST.
 INPUTS:
 DATA: UIC_CUST_IN.
 OUTPUTS:
 DATA: UIC_CUST_WORF.
 REFERRED BY:
 SUBNET: CONTINUE_XMA_C_PROCESS
 SUBNET: PROCESS_XMA_A.
 ALPHA: STORE_UIC_SPT_NR_ANU_SEQ_NR.
 CREATES:
 ENTITY_CLASS: CROSS_REFERENCE_FILE.
 INPUTS:
 DATA: SEQ_NO_IN
 DATA: UIC_SPT_IN.
 OUTPUTS:
 DATA: SEQ_NO_ANU_WORF
 DATA: UIC_SPT_ANU_WORF
 DATA: UIC_SPT_WORF.
 SETS:
 ENTITY_TYPE: MANEUVER_CUSTOMER_B_CARD.
 REFERRED BY:
 SUBNET: PROCESS_XMA_A.

AMX81-085

Figure 3-20 RSL Definition of the ALPHAs in SUBNET: PROCESS_XMA_A

the fact that all relationships are shown whenever an element in the data base is listed.

3.5.7 R NETs as Source of Trouble Reports

The development of R_NETs required nearly half of the labor applied to this effort. A total of 2 R_NETs and 248 SUBNETS were defined using the technique described above. Perhaps it's not surprising, then, to note that nearly all of the discrepancies identified and documented as Trouble Reports were discovered in this phase of the effort. As each DLT was evaluated, it was translated into an R_NET, and logical errors became very apparent. A discussion of Trouble Reports resulting from R_NET development, and from efforts in other phases of the contract is provided in Section 4 of this report.

3.6 DEVELOPMENT OF TRACEABILITY

In a system such as the Standard Army Maintenance System (SAMS) where several levels of requirements and design specifications exist, both upward and downward traceability should exist. This allows verification of performance against the parent requirements and allows an impact analysis to be made in the event that a detailed performance requirement cannot be met, or a change in the system requirement occurs. RSL definitions that are related to traceability are provided in Table 3.8, and their inter-relationships are shown in Figure 3-21.

Table 3.8 RSL Definitions Used in the Development of Traceability

DEFINITION OF ELEMENTS	
ORIGINATING REQUIREMENT	A HIGHER LEVEL OF REQUIREMENTS FROM WHICH LOWER LEVEL REQUIREMENTS, THOSE THAT ARE EXPRESSED IN RSL, ARE TRACEABLE.
SOURCE	SOURCE OR AUXILIARY MATERIAL FOR REQUIREMENTS. IT IS THE ORIGINATING POINT FOR ONE OR MORE ORIGINATING REQUIREMENT, THE DOCUMENTATION OF TRADE-OFF STUDIES, OR THE BACKGROUND MATERIAL FOR REQUIREMENTS ELEMENTS.
DECISION	A CHOICE OF INTERPRETATION THAT HAS BEEN MADE TO ESTABLISH FUNCTIONAL AND/OR PERFORMANCE REQUIREMENTS BASED ON ONE OR MORE ORIGINATING REQUIREMENTS.
DEFINITION OF RELATIONSHIPS	
DOCUMENTED BY (DOCUMENTS)	IDENTIFIES THE ORIGINATING POINT OR PROVIDES AUXILIARY INFORMATION FOR THE ORIGINATING REQUIREMENT.
INCORPORATES (INCORPORATED IN)	INDICATES A HIERARCHICAL RELATIONSHIP BETWEEN ORIGINATING REQUIREMENTS.
TRACES TO (TRACED FROM)	IDENTIFIES THE LOWER LEVEL REQUIREMENTS TO OR FROM WHICH THE ORIGINATING REQUIREMENT OR DECISION HAVE BEEN ALLOCATED OR DERIVED.

ANALYST-008

The SOURCE of the ORIGINATING REQUIREMENTS established in this phase is the text of Chapter 4 of the DFSR. We will use an excerpt from that chapter to illustrate how the ORIGINATING REQUIREMENTS are developed. Refer to the excerpt from Paragraph 4-10 in Figure 3-22 for the following discussion. In reviewing the contents of this text to find elements that

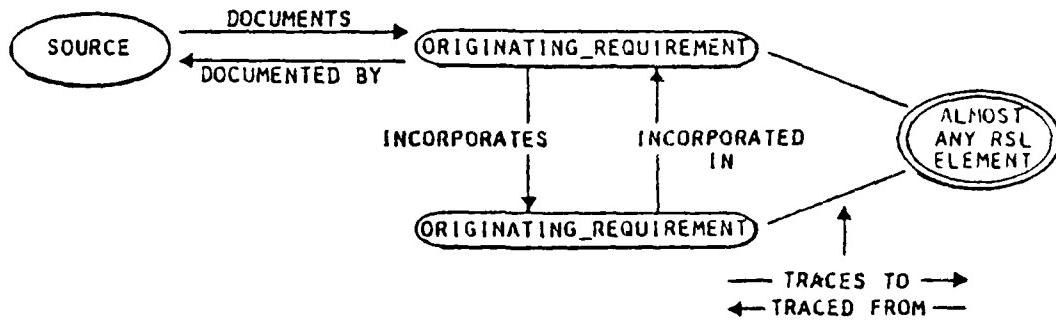


Figure 3-21 Traceability Interrelationships

ANXII-017

4-10. WORK ORDER MANAGEMENT (REAL TIME) PROCESS (FIGURE 3-6-2). The Work Order Management Process is a real-time process that provides the vehicle for placing an item of equipment into the support maintenance shop; for processing and controlling an item of equipment through inspection, repair, and final inspection; and for returning an item of equipment to the customer. This process accepts information from the Maintenance Request/Work Order form by keying changes in work status into the R/W as they occur. This current (real-time) data is thus available for the manager to secure current status of the work orders in the shops for operational management purposes and responding to customer inquiries.

4-10a. When a customer determines that support maintenance is required, a Maintenance Request/Work Order, DA Form XXXX (12 30 4R) is prepared (ref para 4-7, fig 3-5-1) and presented with the equipment or item to be repaired to the support shop. The customer should enter any previously assigned Work Order Number, as in ALT/SRO or Equipment Recall schedules.

4-10b. On receipt of a Maintenance Request/Work Order (hereinafter referred to as Work Order (WO)), the Shop Office Clerk reviews the document to insure that all data are entered and that the customer is valid. If a WO is received from a nonvalid customer, the WO may still be accepted, as in the case of a transient; however, unless an emergency exists, the customer is directed to his usual support unit.

4-10c. The Shop Office Clerk prepares to register the work order into the data base. He keys the WO registration document identifier code (XMA/XMB), when prompted by the processor, the data for work order registration (4-11, fig 3-5-2), are entered.

Figure 3-22 Excerpt for Chapter 4 of the DFSR Showing ORIGINATING-REQUIREMENTS

define a process to be implemented by the data processor, only a general description of the overall Work Order Management Process is found prior to subparagraph 4-10c. Therefore, no information in these portions of paragraph constitutes an ORIGINATING REQUIREMENT.

As indicated by the two blocks outlined in paragraph 4-10c, two ORIGINATING REQUIREMENTS were found in this subparagraph. The first requires the DP to recognize an operator request to initiate the WO Registration process XMA/XMB. The second requirement provides that the DP accept, and presumably store, the DATA for Work Order Registration. The following RSL commands established these as ORIGINATING REQUIREMENTS:

ORIGINATING REQUIREMENT: INITIATE_REAL_TIME_PROCESSING.

DOCUMENTED BY SOURCE: PARA_4_10C.

ORIGINATING REQUIREMENT: PROMPT_OPR_ENTRY.

DOCUMENTED BY SOURCE: PARA_4_10C.

ORIGINATING REQUIREMENT: STORE_XMA_XMB_WO_REG_ENTRY.

DOCUMENTED BY SOURCE: PARA_4_10C.

The definition of processing in the Decision Logic Tables should address these ORIGINATING REQUIREMENTS in some defined process or processes. Any important element of the requirements data base that exist because of these requirements should be TRACED FROM them. We define important elements as:

- SUBSYSTEM
- INPUT_INTERFACE
- OUTPUT_INTERFACE
- MESSAGE
- ENTITY_CLASS
- R_NET
- SUBNET.

Thus, when the requirements data base is complete, each of the above element types should be TRACED FROM one or more ORIGINATING REQUIREMENTS. Conversely, every ORIGINATING REQUIREMENT should show a TRACES TO relationship to one or more of the important elements, defined above.

If the TRACED FROM relationship does not exist for an important element, this suggests that the element was not needed since no

ORIGINATING_REQUIREMENT called for it. And if an ORIGINATING_REQUIREMENT does not trace to any important element, it implies that the software requirements are not complete, and won't be complete until some portion of the requirements statement is prepared to satisfy that ORIGINATING_REQUIREMENT. Thus, at the conclusion of the creation of the requirements data base, RADX is used to identify either of these cases of faulty traceability so it may be addressed by the software engineer.

A total of 232 ORIGINATING_REQUIREMENTS were developed from Chapter 4 of the MOM DFSR. Each of the 501 "important elements" were traced to these ORIGINATING_REQUIREMENTS. As a result of this effort, 167 important elements were not traceable to any ORIGINATING_REQUIREMENT. Similarly, 47 ORIGINATING_REQUIREMENTS did not trace to any data base element. These are the results of the first pass of the RADX traceability check. Although we lacked available computer time to make corrections, our assessment indicates that most of the untraced elements were due to human error in the data base. This further illustrates the fact that although human error can occur with the application of any requirements engineering technique (including SREM), the RADX capability highlights all the errors so that they may be corrected. Consequently, these errors would have been corrected in due course in a normal verification effort.

With an easy RSL extension this traceability can be continued to software modules, test requirements, test cases, etc. Therefore, the data base has continuing utility for support of configuration management during software development and test, and even after the system is fielded.

The perceptive reader will have already recognized one additional valuable benefit of establishing strong traceability beyond that of configuration management support. That is the powerful support of the assessment of changes to the requirements that is possible. Suppose, for example, a SOURCE paragraph was to be deleted as a requirement. RADX would allow the requirements data base to be queried to produce a list of elements that are TRACED from the ORIGINATING_REQUIREMENTS DOCUMENTED by that SOURCE paragraph.

It does not follow that all such elements should be purged from the data base. Rather, all should be reviewed with an eye to other processes that are not to be deleted. Recognition of other involvement can be gained by noting the various relationships of the elements on the list. If, for

example, an ALPHA on an R_NET, which is to be deleted because of the requirements deletion, is also on another R_NET which is not to be deleted, it must be retained. If it was found only on (REFERRED BY) the R_NET to be deleted it could safely be deleted from the data base. Additionally, the DATA and FILE information INPUT TO or OUTPUT FROM that ALPHA would also be examined and, if not used for any other purpose, could also be deleted. Thus, SREM provides the capability of removing unneeded processing when a change occurs, instead of leaving it in place due to the fear of unexpected impact on other processing.

3.7 EVALUATION USING RADX

The Requirements Analysis and Data Extraction (RADX) function built into the REVS software provides a means to accomplish powerful automated checks for the completeness, consistency and traceability of the software requirements entered into the data base as a result of the efforts described in the preceding paragraphs. This is a unique capability, compared to other requirements engineering tools and/or methodologies.

Because much of the results of our effort, including the example of the regeneration of the MOM DFSR specification, utilizes the RADX capability, it is appropriate to describe rather thoroughly how this tool is used. This will assist in understanding later displays of information from the requirements data base and for illustrating the power of RADX in its support of the software requirements engineer.

In this discussion, we will first define the basis for the use of RADX, describe how RADX is used to create sets of interest from the information in the requirements data base, how to list these sets for inspection, and how to produce CALCOMP plots from structures in the data base. With this understanding of how RADX is used, we will describe the steps we took to analyze the adequacy of the requirements data base, to include examples where appropriate for understanding. Finally, we will summarize the problems found using RADX that were reported via Trouble Reports.

3.7.1 The Premise of RADX Use

Conceptually, RADX is built on the premise that if the preceding efforts were accomplished in accordance with the prescribed methodology, certain properties about the elements and their relationships should be true, if the effort is complete, and conversely, certain properties should be absent if it isn't. For example, every MESSAGE defined in the data base as entering the DP system to stimulate some prescribed processing, or as being produced during the processing as a message to be transmitted from the DP system to an outside SUBSYSTEM, must be MADE BY DATA or FILE information. Thus, any MESSAGE that is not MADE BY any DATA or FILE information is a requirements data base error that must be corrected. Once this MESSAGE is identified the requirements engineer will accomplish one of three corrections, namely: 1) determine the DATA and FILE information and ascribe it to the MESSAGE using the relationship: MESSAGE is MADE BY DATA,

or MADE BY FILE, or 2) determine that the MESSAGE is really not required and PURGE it from the data base, or 3) determine that the MESSAGE name is a slight variant of the naming of another MESSAGE in the data base and is, in fact, meant to be the same one. In this latter case, the improperly named MESSAGE is MERGED into the correctly named MESSAGE (thus causing the relationships and attributes of the improperly named MESSAGE to now be ascribed to the correctly named one). In addition, the incorrect MESSAGE name is PURGED from the data base.

3.7.2 RADX Approach

The approach used for RADX analysis is sets analysis. A subset of the information in the requirements data base is defined and a RADX command is input to create that subset. This may be a pre-defined set such as any of the basic elements (e.g., ALL (everything in the data base), MESSAGE, R NET, SUBNET, DATA, etc.), or may be a user-defined subset.

3.7.2.1. Definition of a Set by Relationship Qualification

The user may define a subset through the combination of an element and some relationship or attribute ascribed to that element. If a subset of MESSAGES that are not passed by an INPUT_INTERFACE is desired, the RADX command to establish that set is:

SET UNPASSED MESSAGES = MESSAGE THAT IS NOT PASSED.

In the example, the indicated elements are:

- SET The RADX command to indicate that a new set from the requirements data base is to be established.
 - UNPASSED_MESSAGES An arbitrary set_identifier (name) given to the new_set. Any name may be used but is usually worthwhile to use a meaningful name.
 - = The definition of the set just named will now be defined.
 - MESSAGE A predefined subject SET which initiates definition of the SET: UNPASSED_MESSAGES.

- THAT A legal positive connector which will link some relationship (in this case) or attribute concerning the element: MESSAGE to define the set.
 - IS An optional word which may be used with this positive connector.
 - NOT A legal connector that makes the foregoing connector a negative connector.
 - PASSED A legal relationship for the subject SET: MESSAGE in this set definition.

Thus, this SET is defined as all MESSAGES that currently reside in the requirements data base that are not defined as passing across any INPUT_INTERFACE or OUTPUT_INTERFACE. Note that the object elements INPUT_INTERFACE and OUTPUT_INTERFACE did not have to be named to create this SET. This is because REVS recognizes that the relationship PASSES is legal only when MESSAGE is the subject element of this definition and either INPUT_INTERFACE or OUTPUT_INTERFACE is the object element. Thus, when only the relationship PASSES is used all MESSAGES are included, regardless of whether they are PASSED by the INPUT_INTERFACE or by the OUTPUT INTERFACE.

Of course the object element may be used in the SET definition if appropriate to the user's intent in creating the SET. For example, suppose it was desired to create a SET of just the input MESSAGES. In that case, the RADX command would be:

SET INPUT_MESSAGES = MESSAGE THAT IS PASSED BY INPUT_INTERFACE.
This command would create the desired SET. The SET would not include
MESSAGES PASSED by any OUTPUT_INTERFACE, nor MESSAGES not PASSED by any
interface.

The general case, then, for a relationship qualification is:

SET Set_identifier = Existing_subject_set_identifier
Positive_connector [or Negative_connector]
[MULTIPLE] Relationship_name [Relationship_optional_word]
[Object set identifier].

The bracketed phrases represent optional portions of the SET definition. All of those shown except "MULTIPLE" have been explained. The term

"MULTIPLE" is used when more than one instance of the indicated relationship must exist for an item to be part of the defined SET. For example, the RADX SET command:

```
SET MESSAGES_PASSED_BY_MORE_THAN_ONE_INTERFACE  
      = MESSAGE THAT IS MULTIPLE PASSED.
```

would create a SET of all MESSAGES that are passed by more than one interface (an improper condition).

3.7.2.2 Definition of SETs by Attribute Qualification

SETS may also be created by using attribute names, or by using actual attribute values. In the first case, suppose it was desired to create a SET of R_NETs that did not have the attribute DESCRIPTION. The RADX command would be:

```
SET UNDESCRIBED_NETS = R_NET WITHOUT DESCRIPTION
```

The indicated elements in this example are:

- SET The RADX command to indicate that a new set from the requirements data base is to be established.
- UNDESCRIBED_NETS An arbitrary Set_identifier (name) given to the new set.
- = The set just named will now be defined.
- R_NET A predefined subject SET which initiates definition of the SET: UNDESCRIBED_NETS.
- WITHOUT A legal negative connector.
- DESCRIPTION A predefined legal attribute for the subject SET: R_NET.

The use of the value of an attribute to describe a SET to be created can best be illustrated by another example. Suppose it was desired to know all the data items with the attribute: UNITS which have the value MANHOURS. The appropriate RADX command would be:

```
SET DATA_WITH_UNITS_OF_MANHOURS = DATA WITH UNITS = MANHOURS.
```

-OR-

```
SET DATA_WITH_UNITS_OF_MANHOURS = DATA WITH UNITS MANHOURS.
```

When the Relational_operator in the SET definition is "=", it may be omitted, since that is the default value. The general case for attribute qualification is:

```
SET Set_identifier = Existing_subject_set_identifier  
    Positive_connector [or Negative_connector]  
    Attribute_name [Relational_operator]  
    [Attribute_value].
```

The legal relational operators are outlined in Table 3.9.

Table 3.9 RADX Relational Operators

RELATION OPERATOR	MEANING
>	GREATER THAN
>=	GREATER THAN OR EQUAL
<>	NOT EQUAL
=	EQUAL
<=	LESS THAN OR EQUAL
<	LESS THAN

The value that is specified for Attribute_value can be an integer, a real number, a value name, or a text string that is not longer than 60 characters. The relational operators = and <> are the only ones that are legal if the value is specified as a text string or as a value name in the set definition.

The members of any new SET are those members of the existing subject SET that satisfy the relationship or attribute criterion in the manner indicated by the connector. When a positive connector is used, the members of the new SET are those in the existing subject SET which have the stated relationship or attribute criterion. If a negative connector is used, then the members of the new SET are those in the subject SET that do not have the stated relationship or attribute criterion. A variety of terms are allowed to be used as positive and negative connectors to increase the readability of RADX statements. The list of legal connectors useable in RADX set definition are shown on Table 3.10.

Table 3.10 RADX Positive and Negative Connectors

POSITIVE_CONNECTOR	NEGATIVE CONNECTORS
WITH WHERE WHICH WHICH IS IN FROM SUCH SUCH THAT THAT THIS IS BY	POSITIVE_CONNECTOR NO POSITIVE_CONNECTOR NOT WITHOUT

ANX81-011

3.7.2.3 Definition of a Set by Enumeration and Combination

Whereas the previous examples illustrated SET definition in terms of a single existing subject SET, RADX also provides the means to define new SETs via use of more than one existing SET. This can be accomplished by enumeration or by combination of existing SETs.

In the enumeration approach, the members of a new SET can be defined as those contained in another existing SET or as the union of two or more existing SETs.

The statements given below demonstrate the use of this technique for defining SETs.

SET A = ALPHA, FILE, INPUT_INTERFACE.

SET B = SET A, DATA.

In the first statement, SET A will contain all the elements in the data base that are members of the predefined element type SETs ALPHA, FILE or INPUT_INTERFACE. The SET B will contain elements that are members of the user defined SET A plus the predefined element DATA.

Defining Sets by Combination

A set can be defined as the logical combination of two existing independent sets by a statement using the following syntax:

```
SET Set_identifier = Existing_first_independent_set_identifier
    Combination_operator Existing_second_independent_set_
        identifier
```

The combination_operators are:

- AND - SET intersection. The members of the new SET are those that are members of both the first independent SET and the second independent SET.
- OR - SET union. The members of the new SET are those that are members of either the first independent SET or the second independent SET.
- MINUS - SET difference. The members of the new SET are those that are members of the first independent SET, but not the second independent SET.

Examples of SET definition by combination using these operators follow:

- SET ALPHA DATA = ALPHA OR DATA. This combines all ALPHAs and DATA into a single SET. This provides the same result as the following SET definition by enumeration:
SET ALPHA_DATA = ALPHA, DATA
- SET NETS IN X = R NET and X. Here, X is a user-defined SET which may include several different predefined or user-defined SETs. If it includes R_NETs, these R_NETs will now exist as the SET: NETS IN X. If there are no R_NETs in SET X, NETS_IN_X will be an empty set.
- SET ALL EXCEPT DECISION = ALL MINUS DECISION. This removes the predefined SET: DECISION from the total existing requirements data base. The set of remaining elements are now defined as the SET: ALL_EXCEPT_DECISION.

3.7.2.4 Defining SETs By Structure Qualification

Implicit relationships between structures and elements used in structures may be used for defining a new SET of elements that have, or do not have, certain structural characteristics. These implicit relationships are named REFERS and REFERRED. They cannot be explicitly input through the RSL translator but they are implicitly defined when a structure is entered into the requirements data base.

The REFERS relationship exists between an element with a structure and the elements used on the structure. The REFERRED relationship is the

complement of the REFERS relationship. These implicit relationships are used in the same manner as RSL relationships are used to define a new SET by relationship qualification. The following examples illustrate different uses of this statement.

SET R_NET_NO_STRUCTURE = R_NET WITHOUT REFERS.

SET R_NET_USING_UPDATE_STATE = R_NET WHICH REFERS TO UPDATE_STATE.

SET ALPHAS_NOT_USED = ALPHA THAT IS NOT REFERRED.

SET ALL_NEEDED_BY_R_NET_RADAR_SUMMARY = ALL THAT IS REFERRED TO BY RADAR_SUMMARY.

3.7.2.5 Defining Sets by Hierarchy

There are several hierarchies that exist in the definition of RSL such as data hierarchies and structure hierarchies that can be identified for use as a "road map" to trace through the requirements data base for the purpose of defining a SET or determining the order in which to extract and display information. A RADX HIERARCHY can be defined as follows:

```
HIERACHY [OR HIER] New_hierarchy_name =
  (Existing_subject_set_identifier Relationship_name
   [Relationship_optional_word]
   Existing_object_set_identifier)n
```

The symbol ()ⁿ indicates that the type of relational statements between the parentheses () may be repeated any number of times. In the above definition, New_hierarchy_name is a unique name that will be used to reference the HIERARCHY. The set_identifiers designate SETs that must be defined before the HIERARCHY is defined, and Relationship_name is any RSL relationship or an implicit relation (REFERS or REFERRED).

For example, Figure 3-23, illustrates an RSL information hierarchy that exists in the requirements data base. The nodes in the graph represent SETs (in this case predefined element type SETs) and the branches

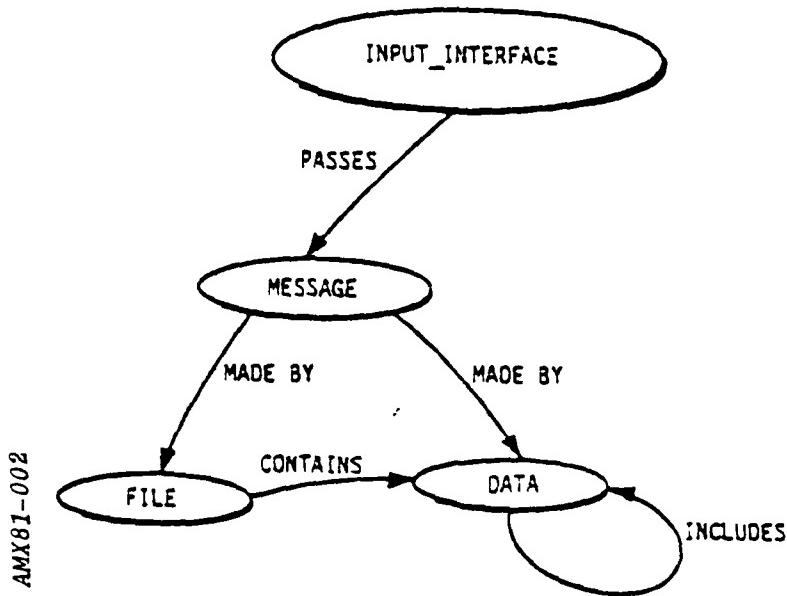


Figure 3-23 Hierarchy for INPUT INTERFACE

represent binding relationships between the SETs. This HIERARCHY can be named, say INFO_SOURCE, and input under RADX for use by defining the connectivity of the graph as follows:

```

HIER INFO_INTERFACE = INPUT_INTERFACE PASSES MESSAGE;
                      MESSAGE MADE BY FILE;
                      MESSAGE MADE BY DATA;
                      FILE CONTAINS DATA;
                      DATA INCLUDES DATA.
  
```

It may then be LISTed. Examples of LISTed HIERARCHIES can be found in Figures 3-2, 3-3, and others provided earlier.

It often is appropriate to create a SET of all elements traversed via the HIERARCHY. If it was desired to create a SET called INFO_SOURCE for all the elements in the HIERARCHY: INFO_INTERFACE, it would be accomplished as follows:

3.7.2.6 Listing RADX Sets

Any predefined or user-defined set created under RADX procedures can be listed by the simple command:

LIST Set_identifier

For example, the command:

LIST ALL or LIST ANY

will cause all elements defined in the requirements data base to be listed.

The command:

LIST DATA

will cause only the DATA in the requirements data base to be listed. The command to

LIST X

(where X is a user-defined set) will cause the user-defined set to be printed. In addition, if it were desired, the output could be punched cards, rather than a printout, by substituting the command: PUNCH for LIST.

3.7.2.7 Controlling the Listing Format

When RADX is initially activated, all elements in a RADX listing will include all information (relationships, attributes, structures) known about each. The amount of information to be listed can be controlled by using the RADX command called APPEND.

The APPEND command is used to specify the associated attributes, relationships, and structures that should be displayed along with the display of an element. The syntax of the statement is:

APPEND Element_type_identifier (Append_item)₁ⁿ.

The)₁ⁿ indicates that there may be a multiple list of Append_items for a particular Element_type_identifier. When this is true, the information is printed in the order in which the Append_item is listed.

In the above statement, Element_type_identifier is an RSL element type name, such as MESSAGE, DATA, etc., or the keyword ANY or ALL, and indicates the element type or element types to which the Append_item applies. When ALL or ANY is specified, the Append_item is applied to all element types in the requirements data base. A list of legal Append_items and the information that each causes to be appended to an Element_type_identifier is shown in Table 3.11.

Table 3.11 Append_Item List

RELATION_NAME	A PARTICULAR RSL RELATIONSHIP.
ATTRIBUTE_NAME	A PARTICULAR RSL ATTRIBUTE.
REFERS	ELEMENTS THAT APPEAR ON THE STRUCTURE OF THE SUBJECT ELEMENT
REFERRED	ELEMENTS WITH STRUCTURES THAT USE THE SUBJECT ELEMENT.
ALL	ALL ATTRIBUTES IN ALPHABETICAL ORDER, FOLLOWED BY ALL PRIMARY RELATIONSHIPS IN ALPHABETICAL ORDER, FOLLOWED BY REFERS, FOLLOWED ALPHABETICALLY BY ALL COMPLEMENTARY RELATIONSHIPS, FOLLOWED BY REFERRED, AND FINALLY THE ELEMENT STRUCTURE OR PATH.
NONE	NO ASSOCIATED INFORMATION.
STRUCTURE	R_NET, SUBNET, OR VALIDATION_PATH STRUCTURE.
ATTRIBUTE	ALL ATTRIBUTES IN ALPHABETICAL ORDER.
RELATION RELATIONSHIP }	ALL PRIMARY RELATIONSHIPS IN ALPHABETICAL ORDER FOLLOWED BY ALL COMPLEMENTARY RELATIONSHIPS IN ALPHABETICAL ORDER.
PRIMARY	ALL PRIMARY RELATIONSHIPS IN ALPHABETICAL ORDER.
COMPLEMENTARY	ALL COMPLEMENTARY RELATIONSHIPS IN ALPHABETICAL ORDER.

3.7.2.8 Displaying Structures

The command: PLOT provides a means for attaining a CALCOMP plot for any structure which has been entered into the data base. The general form of the command is:

PLOT Existing_structure_element Plot_size.

The term Existing_structure_element means R_NET, SUBNET, or VALIDATION_PATH. The term Plot_size refers to the desired size of the requested plot. This is written as:

WIDTH = Width_value, HEIGHT = Height_value.

The Width_value and Height_value are stated in inches and as a real or integer number up to 50 and 29, respectively. The default values (if no values are provided) are 8 and 10, respectively.

Figure 3-17, shown earlier, illustrated a hand drawn SUBNET whose structure was translated into RSL and placed in the requirements data base. Figures 3-24 and 3-25 provide its CALCOMP equivalent from the data base. Note that the resulting plot is actually two pages of output. The plot of the SUBNET is shown Figure 3-24. The title of the SUBNET is shown at the

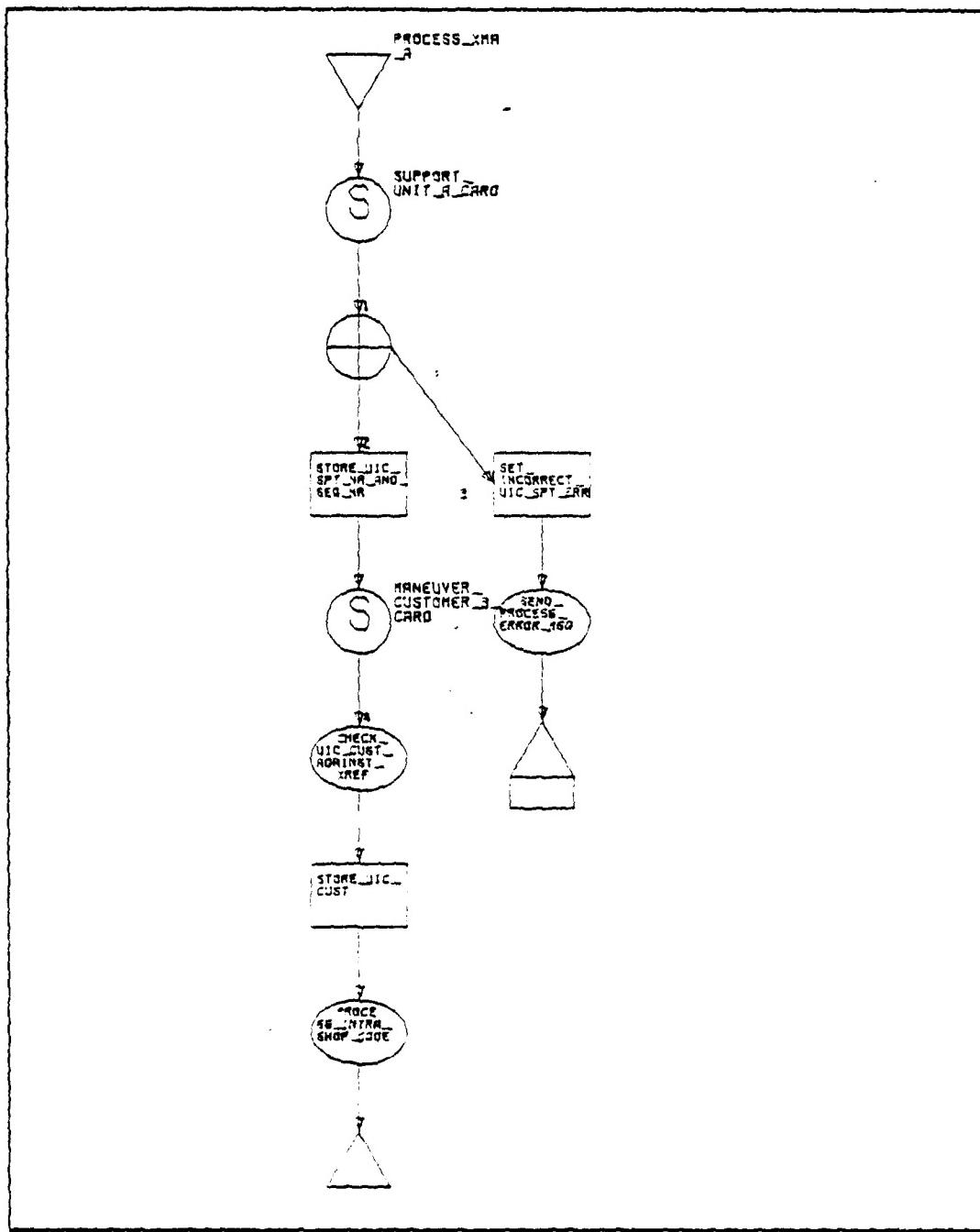


Figure 3-24 CALCOMP Plot of SUBNET: PROCESS_XMA_A

PROCESS_XMA_A STRUCTURE LEGEND		
NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(UIC_SPT_CRF=UIC_SPT_IN)
2		(FOUND)
3		OTHERWISE
4		((UIC_SPT_CRF=UIC_SPT_IN) AND (UIC_SUST_CRF_B=UIC_SUST_IN))

Figure 3-25 Conditional Expressions for the CALCOMP Plot for SUBNET:
PROCESS_XMA_A

top of the plot. The plot looks similar to its hand-drawn equivalent in Figure 3-17, except that the conditional expressions found on the branches of the hand-drawn SUBNET are replaced by numbers on the CALCOMP version. The second sheet of the CALCOMP plot (Figure 3-25) defines the conditional expressions represented by these numbers and also provides any comments that may have been entered for the structure. This two-page approach was selected due to its ease of implementation during development, and has not been subsequently improved. This feature, and certain other awkward aspects of the current CALCOMP plot capability represent one of the REVS areas that deserves improvement, such as the inability to be read when there is a large number of nodes on the structure.

3.7.2.9 Combined Static RADX Tests

In order to reduce the load on the software engineer, and to assure a complete RADX check is made of the requirements data base, a set of standard RADX commands has been derived for all appropriate RADX tests. These have been organized in groups related to specific phases of the methodology and stored as ADDFILEs for access after each phase is complete. Such tests exist for Phases 1, 2, 3, 4, and 6. Phases 5 and 7 are simulation phases, and no static RADX tests are used in these phases. When used, any LISTed SETs indicate failure of the software engineer to consistently, completely, and unambiguously describe the software requirement in accordance with the SREM procedures. Attention should be given to the items listed to correct the indicated problems. A comment is provided for each LIST command to describe what error the members of the SET possess, or to describe the intent of the RADX test whose result is being listed. A listing of the kinds of tests performed for each phase of SREM are provided in Tables 3-12 through 3-16 for Phases 1, 2, 3, 4, and 6, respectively.

Several of the combined RADX tests appropriate for the MOM DFSR effort were administered to the requirements data base. The results of these tests are provided in Appendix C. Normally, each of the indicated problems would be corrected, and many of the answers needed for this correction would come from the user. However, under this contract, there is no user, and therefore, no correction of the data base has been attempted.

Table 3.12 SREM Phase 1 RADX Checks

<u>ENTITY_CLASS</u> <ul style="list-style-type: none"> • NOT CREATED • NOT DESTROYED • WITHOUT ENTITY_TYPE
<u>ENTITY_TYPE</u> <ul style="list-style-type: none"> • NOT SET • NOT IN AN ENTITY_CLASS • IN MORE THAN ONE ENTITY_CLASS
<u>MESSAGE</u> <ul style="list-style-type: none"> • OUTPUT_MESSAGES NOT FORMED

Table 3.13 SREM Phase 2 RADX Checks

<u>PROVIDES FOR VISUAL CHECK</u> <ul style="list-style-type: none"> • ENTITY_CLASS HIERARCHIES • FREE STANDING FILES • FREE STANDING DATA • INPUT_INTERFACE HIERARCHIES • OUTPUT_INTERFACE HIERARCHIES
<u>PROBLEM REPORTS</u> <ul style="list-style-type: none"> • OPEN PROBLEMS • NOT TRACED FROM ORIGINATING_REQUIREMENT OR ANOTHER DECISION • NOT TRACED TO ELEMENTS IT WILL IMPACT
<u>SUBSYSTEMS</u> <ul style="list-style-type: none"> • UNCONNECTED
<u>INTERFACES</u> <ul style="list-style-type: none"> • UNCONNECTED • CONNECTED TO MORE THAN ONE SUBSYSTEM • PASSES NO MESSAGES • AN INPUT-INTERFACE DOESN'T ENABLE R-NET
<u>MESSAGES</u> <ul style="list-style-type: none"> • WITHOUT DATA OR FILES • NOT PASSED BY AN INTERFACE • PASSED BY MORE THAN ONE INTERFACE
<u>R-NETS</u> <ul style="list-style-type: none"> • NOT ENABLED BY AN INPUT_INTERFACE OR EVENT • MULTI_ENABLED • ENABLED BY INPUT_INTERFACE NOT ON STRUCTURE
<u>EVENTS</u> <ul style="list-style-type: none"> • DOESN'T ENABLE AN R-NET • DELAYED BY DATA NOT AT LOWEST LEVEL • DELAYED BY MORE THAN ONE DATA ITEM
<u>STRUCTURES</u> <ul style="list-style-type: none"> • AN ALPHA, SUBNET, EVENT, VALIDATION_POINT, INPUT_INTERFACE, OR OUTPUT_INTERFACE DOES NOT APPEAR ON A STRUCTURE • LOGIC NOT DEVELOPED FOR DEFINED STRUCTURE
<u>ENTITIES</u> <ul style="list-style-type: none"> • ENTITY_CLASS (OR ONE OF ITS ENTITY_TYPES) NEVER SELECTED ON A STRUCTURE • ENTITY_TYPE DOES NOT CONTAIN ANY DATA OR FILES

Table 3.14 SREM Phase 3 RADX Checks

<u>FILE ORDERING</u>
• MULTI_ORDERED FILES
• FILE ORDERING DATA NOT IN A FILE
• NON_LOWEST LEVEL ORDERING DATA
<u>FILE CONSTRUCTION</u>
• EMPTY_FILES
<u>DATA</u>
• NEITHER A SINK NOR A SOURCE
• SOURCE, BUT NO SINK
• SINK BUT NO SOURCE
• MISSING RANGES FOR ENUMERATED DATA
• RANGE FOR TYPES OTHER THAN ENUMERATION
• NO TYPE
• USE INFORMATION (NEEDED FOR SIMULATIONS)
• DATA IN MORE THAN ONE FILE
• DATA IN MORE THAN ONE ENTITY_CLASS
• DATA IN BOTH AN ENTITY_CLASS AND ENTITY_TYPE
• DATA IN BOTH AN ENTITY AND IN A FILE
• DATA IN BOTH AN ENTITY AND IN A MESSAGE
• DATA IN BOTH A MESSAGE AND A FILE
• LOCAL DATA IN ENTITIES
• GLOBAL DATA IN MESSAGES
• LOCAL DATA IN A GLOBAL FILE
• GLOBAL DATA IN A LOCAL FILE

ANXII-020

Table 3.15 SREM Phase 4 RADX Checks

<u>TRACEABILITY</u>
• ORIGINATING_REQUIREMENTS THAT DON'T TRACE TO OTHER ELEMENTS
• DECISIONS THAT DON'T TRACE TO OTHER ELEMENTS
• SOURCES THAT WEREN'T USED
<u>AUTHORSHIP</u>
• UNAUTHORED DATA BASE ELEMENTS
<u>DESCRIPTIONS</u>
• UNDESCRIPTED NETS

ANXII-021

Table 3.16 SRM Phase 6 RADX Checks

<u>VALIDATION_POINTS</u>
• NOT PLACED ON A NET
• PLACED ON MORE THAN ONE NET
• DOESN'T RECORD ANY DATA OR FILES FOR TEST
<u>PERFORMANCE_REQUIREMENTS</u>
• HAS NO TEST WRITTEN FOR IT
• HAS NO VALIDATION PATH FOR IT
• NOT TRACED FROM AN ORIGINATING_REQUIREMENT, A DECISION, OR A SOURCE
<u>VALIDATION_PATHS</u>
• NOT CONSTRAINED BY A PERFORMANCE_REQUIREMENT
<u>PROVIDES FOR VISUAL CHECK</u>
• PERFORMANCE_REQUIREMENTS WITHOUT CONSTRAINTS TO CHECK AGAINST VALIDATION_PATHS NOT CONSTRAINED
• UNCONSTRAINED VALIDATION PATHS TO SEE IF A PERFORMANCE_REQUIREMENT SHOULD BE WRITTEN TO CONSTRAIN IT
• SHOWS HOW MANY PERFORMANCE_REQUIREMENTS EACH VALIDATION_PATH IS CONSTRAINED BY
• SHOWS HOW MANY VALIDATION_PATHS ARE CONSTRAINED BY EACH PERFORMANCE REQUIREMENT.

AMX81-022

3.7.2.10 Data Flow Analysis

The final static RADX test of the data base is the data-flow analysis. This is accomplished by the command:

ANALYZE DATA_FLOW Structure_set_identifier.

where Structure_set_identifier is a class of structures (such as R_NET), or is the name assigned for the particular R_NET or SUBNET whose analysis is desired.

The basic data-flow tests of interest are:

- Loop Detection: Identifies any loops that may have been inadvertently introduced due to faulty SUBNET referencing or a recursive DATA definition via the INCLUDES relationship.
- LOCALITY Attribute Test: Checks the LOCALITY attribute for DATA and FILEs used or produced in the R_NET to determine whether LOCAL or GLOBAL. It then assures that appropriate use is made in accordance with the assigned

LOCALITY. For example, only LOCAL DATA and FILEs may be in a MESSAGE and only GLOBAL DATA and FILEs may be ASSOCIATED with an ENTITY_CLASS or ENTITY_TYPE.

- Membership Test: Identifies inadvertent inclusion of a DATA item in more than one repetitive data set (ENTITY_CLASS, ENTITY_TYPE, or FILE).
- Conditional Branching Tests: Checks for ambiguous or incomplete statement of branching condition.
- Net Structure Test: Checks for SUBNETS that are REFERRED, but which do not have processing logic defined, and for incorrect structure logic caused by improper rejoins after OR or AND node branching.
- Information Assignment/Usage Tests: Checks for information used without a source, and information assigned (produced) that is not used.
- Ambiguous Flow Test: Checks to see if the change in the sequence of processing of paths initiated by an AND node could cause the source of information for any following R_NET node to change.

After listing the flow of DATA within the R_NET, the MESSAGES input to the R_NET and produced by it (with the FILE and DATA that MAKES them), any errors detected by the tests described above are listed. The locations of errors in the R_NET are pin-pointed by a list of walk back information which shows the preceding nodes in the order traversed from each error node back to the top of the R_NET. An example of a data flow analysis for the XMA Real-Time Process will be found in Appendix C.

3.8 ANALYSIS OF THE SRM APPLICATION EFFORT

During the period of performance two requirements engineers worked full time on this effort, two others worked part time, and supervision added an equivalent effort of just over one-half a person. Over the approximately 6-1/2 months of technical activity this was the equivalent of just under 2-2/4 requirements engineers. The SRM engineering effort applied to evaluation of the MOM DFSR totals 2660 man-hours. Engineering man-hours were allocated to 16 identifiable tasks as shown in Table 3.17. Since the R_Net is the focal point of SRM activity, much time was spent on it for the MOM DFSR. It has, historically, been one of the most time consuming activities. A total of 1152 man-hours have been applied to date to defining R_Nets. That is 43.1 percent of the total engineering time applied in this effort.

Table 3.17 SRM Task/Time Allocation

	Hours Applied	% OF TOTAL HOURS APPLIED
SPECIFICATION RESEARCH	23	.8%
RECORD PROBLEMS	146	5.5%
DEFINE SUBSYSTEMS	20	.8%
DEFINE INTERFACES	26	.9%
DEFINE MESSAGES	54	2.0%
DEFINE MESSAGE CONTENTS	196	7.4%
DEFINE R_NETS	1152	43.1%
ENTITY CLASS	41	1.6%
ENTITY TYPE	6	.2%
ENTITY DATA DETAIL	6	.2%
ESTABLISH TRACEABILITY	149	5.7%
STATIC RAOX	166	6.0%
REVISE DATA BASE	188	7.1%
REQUIREMENTS REGENERATION	142	5.3%
SRM EVALUATION	25	.9%
REVIEW & COORDINATION	172	6.5%
FINAL REPORT	148	5.7%

As stated earlier in this report, R_Nets are developed to describe the stimulus/response relationship required of software to process each of the input messages. Because of the unique view of processing provided by this technique, most of the specific problems in the software specification were found during the process of defining R_Nets. Those problems were documented by Trouble Reports, the time for which is reported under Record

Problems. A total of 146 man-hours, or 5.5 percent of the effort, was applied to this task.

Prior to the definition of R_NETs, some preliminary tasks had to occur. These tasks included Specification Research, Definition of SUBSYSTEMs, Definition of Interfaces, GLOBAL DATA Definition (ENTITY_CLASSES, ENTITY_TYPES, and ASSOCIATED DATA detailing). A total of 122 man-hours were applied to these tasks, or 4.5 percent of total man-hours applied. Another preliminary effort included defining MESSAGES, MESSAGE contents, and establishing traceability. In all the preliminary efforts, translation and entry of the defined elements and their relationships into the requirements data base was accomplished concurrently, and is included in the totals. These tasks required 399 man-hours, or 15 percent of total man-hours applied. RADX testing, data base correction, and the regeneration of requirements utilized 496 man-hours, or 18.7 percent of the total. The man-hours applied to administrative type tasks, such as SREM Evaluation, Review and Coordination, and Final Report preparation totalled 345 or 13 percent of total man-hours applied.

Our past experience indicates that the best estimating relationship for determining the amount of effort needed for a SREM application is based on the number of input MESSAGES and EVENTS that stimulate the R_NETs that must be defined. In this effort, there have been 62 input MESSAGES and no EVENTS. A total 2660 man-hours have been applied to the effort, thus averaging 42.9 man-hours per stimulus. Previous experience indicates that approximately 40 man-hours per stimulus is typical, which is comparable with the MOM DFSR effort.

The average is, however, somewhat understated because of certain peculiarities we experienced, although they probably are typical of information systems of this type. Some of the characteristics of the software specification that differ in this application compared to most of our previous applications are:

- Larger set of input and output MESSAGES.
- Larger GLOBAL data base as reflected in the number of ENTITY_CLASSES defined.
- Logical strings of operator/data processor interactions (e.g., prompt, operator response, error message and reprompt, operator response, next prompt, etc.).

- Considerable introduction of design into the requirement, particularly the heavy use of pre-sorting.

SREM was designed to develop and express the logical functional and performance requirement, and not to express actual implementation. To the extent that such implementations are introduced, SREM application efforts increase. Because of implementation in the MOM DFSR, and the other problems, discussed earlier, approximately half of our effort was at a level of summary higher than normal. Although this probably resulted in finding fewer deficiencies in those areas where the approach was used, it did allow us to complete the requirements data base so that important RADX tests could be made. We estimate that, had the engineering assessment been totally applied at the normal level of effort, the application time probably would have increased about 25 percent per input MESSAGE.

If we had not taken the generic approach to the real-time input of information, rather than treating each of the approximately 570 data items that could be entered as separate MESSAGES, application time to define this processing would have been significantly larger. We believe the impact of processing every data item as a separate MESSAGE, as a literal interpretation of this specification would have increased the application effort by about 300 hours. This would amount to slightly more than double the overall effort; under this contract, about 36 man-months.

For specification the size of the MOM DFSR three man-years of effort seems reasonable, when compared to the advantages that will result from having a complete, consistent, and unambiguous specification from which to re-accomplish software design. We believe that much more than this amount of effort and cost would be saved over the remaining life cycle of the development of this software package. An even more positive savings would have resulted if the software specification had been originally developed with SREM. For more discussions of software development costs and the impact of SREM in reducing them, see Section 5.

4.0 RESULTS OF SREM APPLICATION

In preceding sections we have described the basic components of SREM, illustrated how we defined pertinent information to create the requirements data base for the MOM DFSR, and described how we evaluated the data base using the automated RADX capability. During those activities, identified deficiencies in the specification were documented in Trouble Reports and entered into the data base. The purpose of this section is to describe the results of this effort which (for a verification effort of this type) primarily are the documented deficiencies. An added effort, described herein, is the description of the regeneration of the requirements from the REVS data base.

4.1 DESCRIPTION OF TROUBLE REPORTS

The attainment of good software requirements is not easy. Even though the desirable characteristics of a good specification are well understood, capturing those qualities in the production of software specifications has proved elusive. The major qualities that are sought were described in Paragraph 2.2. The development of SREM had the goal of attaining these qualities, when used to develop the original software requirements. It has also proven its capability to identify the lack of these qualities within existing software specifications in its verification role.

Recognized deficiencies in the MOM DFSR were documented using a Trouble Report form as a worksheet, for eventual entry into the data base. Normally, these Trouble Reports would have been submitted to the organization which had developed the specification to attain their response concerning corrective action. In this effort, no such organization existed and, therefore, the MOM DFSR has not been totally corrected. Rather, the deficiencies have simply been recorded to evidence the results of the verification effort.

4.1.1 Trouble Report Format

The Trouble Report form (Figure 4-1) is an AIRMICS version of a standard form that was designed for the interaction between the developer and verifier. Problems found by the verifier are described as completely as possible in the "PROBLEM" block so as to assure understanding by the developer. If any alternatives appear appropriate for solving the PROBLEM, they are included in the "ALTERNATIVES" block by the verifier. The remainder of the header blocks are also completed by the verifier and the Trouble Report is forwarded to the developer. His choice of alternatives or other solution is reported back to the verifier, usually using the "CHOICE" block for his reply. Since we had no developer responses, we filled in the "CHOICE" block on each Trouble Report, wherein we suggested the action needed to correct the stated PROBLEM.

In order to develop statistics on the kinds of deficiencies being identified, a "CATEGORY OF PROBLEM" block was provided for five specific categories, plus an "OTHER" category. The definitions we applied in determining these deficiency categories are as follows:

TROUBLE REPORT NR	DATE PREPARED	DATE CLOSED
SOURCE OF TROUBLE REPORT	<input type="checkbox"/> TM 38-L71-2: OFSR SAMS 1 (MOM) PAGE NR _____ <input type="checkbox"/> TM 38-L72-2: OFSR SAMS 2 (MPOM) PAGE NR. _____	
	TABLE NR _____	
	PARAGRAPH NR _____	
	FIGURE NR _____	
PROBLEM	DECISION:	
	TRACES TO:	
	TRACED FROM:	
	PROBLEM:	CATEGORY OF PROBLEM <input type="checkbox"/> AMBIGUOUS <input type="checkbox"/> MISSING <input type="checkbox"/> INCONSISTENT <input type="checkbox"/> INCOMPLETE <input type="checkbox"/> ILLOGICAL <input type="checkbox"/> OTHER: _____ _____ _____
ASSUMED PROBLEM SOLUTION (CHOICE) OR ALTERNATIVES	ALTERNATIVES:	
	CHOICE:	
TROUBLE REPORT PREPARED BY		DATE ENTERED IN DATA BASE
AIRCRAFTS RESPONSE	<input type="checkbox"/> PROVIDED FOR INFORMATION ONLY <input type="checkbox"/> RESPONSE REQUESTED	
AIRCRAFTS RESPONSE BY	DATE OF RESPONSE	

Figure 4-1 AIRMICS Trouble Report Form

- AMBIGUOUS: Processing requirements from flowcharts, decision logic tables, and processing subparagraphs present unclear intentions due to vague descriptions of logic paths, unclear naming or use of data, or descriptions that defied determination of the true intent of described processing.
- MISSING: Data that is defined as being used or produced during processing, but is missing from Annex C of the DFSR, as well as steps or tables referred to in the DLTs, but which were actually missing. Certain RADX-discovered problems also fall in this category.
- INCONSISTENT: Data names that are different for identical data items, same data name used for different data items, and different data having the same (and therefore, ambiguous) names. Also included are processing steps that are inconsistent between DLTs.
- INCOMPLETE: Processing requirements from DLTs that omit processing logic and/or other required information. Certain RADX discovered problems also fall in this category.
- ILLOGICAL: Processing requirements from DLTs that present processing steps in an illogical order, such that the intended processing cannot be attained.

4.1.2 Entry of Trouble Reports into the Requirements Data Base

When completed, the Trouble Report information was translated from the forms into RSL and entered into the requirements data base. This was accomplished by using the basic RSL element: DECISION. DECISION has predefined attributes and relationships that match major portions of the Trouble Report. These include:

- Relationships:
 - TRACES TO (any RSL Element)
 - TRACED FROM ORIGINATING REQUIREMENT
- Attributes:
 - PROBLEM
 - ALTERNATIVE
 - CHOICE
 - TROUBLE REPORT PREPARED BY (called ENTERED_BY in RSL).

Because RSL is easily extensible, other items on the Trouble Report were created as new RSL elements with new relationships to DECISION, or as new attributes for DECISION. These included the following items:

- Relationships:
 - SHOWN_ON REF_LOCATION (Page Number)
 - IDENTIFIED_BY TROUBLE_REPT_NR
- Attributes:
 - DATE_PREPARED
 - CATEGORY_OF_PROBLEM (AMBIGUOUS, MISSING, ETC.)
 - DATE_CLOSED.

The extension of the RSL to allow the above changes to translate successfully for acceptance and storage is accomplished quite easily. The necessary RSL commands to accomplish this extension are shown in Figure 4-2.

```
ELEMENT_TYPE: TROUBLE_REPT_NR (* *).
RELATION: IDENTIFIES (* *).
COMPLEMENTARY RELATION: IDENTIFIED_BY.
SUBJECT ELEMENT_TYPE: TROUBLE_REPT_NR.
OBJECT ELEMENT_TYPE: DECISION.
ATTRIBUTE: CATEGORY_OF_PROBLEM (* *).
    APPLICABLE: DECISION.
    VALUE: AMBIGUOUS.
    VALUE: MISSING.
    VALUE: INCONSISTENT.
    VALUE: INCOMPLETE.
    VALUE: ILLOGICAL.
    VALUE: OTHER.
ATTRIBUTE: DATE_PREPARED (* *).
    APPLICABLE: DECISION.
    VALUE: TEXT.
ATTRIBUTE: DATE_CLOSED (* *).
    APPLICABLE: DECISION.
    VALUE: TEXT.
ELEMENT_TYPE: REF_LOCATION (* *).
RELATION SHOWS (* *).
COMPLEMENTARY RELATION: SHOWN_ON.
SUBJECT REF_LOCATION.
OBJECT DECISION.

80-14144
```

Figure 4-2 RSL Extensions to Support Trouble Report Entries

4.1.3 RADX Support of Management Review of Trouble Report

RADX was used throughout this project to assure completeness of the Trouble Reports in the data base. The kinds of checks made by RADX included the following:

- Missing TR numbers.
- Missing PROBLEM statements.
- Missing page number source.
- Missing traceability to ORIGINATING REQUIREMENTS.
- Missing CATEGORY_OF_PROBLEM.
- Missing CHOICE.

The RADX commands necessary to automatically check through the large number of Trouble Reports for the kind of problems outlined above is shown in Figure 4-3, and is in the same order as the above list. Wherever the SET COUNT is zero, there are no problems of that type. If the SET COUNT is greater than zero, a command to LIST the SET will provide the names of the DECISIONS which are deficient in the way defined in the SET so that the missing information can be provided.

A critical management function is control. The manager needs tools that he can easily use to control the progress of his project. Proper utilization of the RADX function, such as shown here, helps the manager identify the problems he is interested in so that he may give them the proper attention.

RADX COMMANDS
SET NO_ID_NH = DECISION WITHOUT IDENTIFIED_BY,

SET COUNT = 0

RADX COMMANDS
SET NO_PROBLEM = DECISION WITHOUT PROBLEM.

SET COUNT = 0

RADX COMMANDS
SET NO_SOURCE = DECISION WITHOUT DOCUMENTED_BY.

SET COUNT = 0

RADX COMMANDS
SET NO_TRACEFROM = DECISION WITHOUT TRACED_FROM.

SET COUNT = 19 ;

RADX COMMANDS
SET NO_CATEGORY = DECISION WITHOUT CATEGORY_OF_PROBLEM.

SET COUNT = 0

RADX COMMANDS
SET NO_CHOICE = DECISION WITHOUT CHOICE.

SET COUNT = 0

AMX01-038

Figure 4-3 RADX Checks for Incomplete Trouble Reports

4.2 EVALUATION OF DISCREPANCIES

The size and scope of MOM DFSR, along with its related annexes, suggest a significant effort was originally involved in its development. A strong hierarchy of traceability is communicated throughout the DFSR and a thorough discussion of not only what processing was required to be accomplished, but also how that processing fit into the overall concept of operations was provided. In addition, attempts were clearly made to assure consistency in data naming among the various documents that compose the DFSR. In addition, we found that the Decision Logic Tables (DLTs) used on the DFSR were an excellent approach toward clearly defining the processing required.

4.2.1 DLT Considerations

As with all other efforts of this size, human frailty intrudes to a maximum degree. We observed, for example, different levels of quality (read detail) among the DLTs such that we could almost group them by author. The very fact that the DLTs provide considerable detail increased the opportunity for more errors to occur. Because there probably were no automated tools to assist in the developer's verification of the completeness and correctness of the DLTs, the original review and verification was by other humans with the same set of frailties as the DLT authors.

One of the difficulties often found in the DLTs was the ambiguity in the description of the processing step. For example:

Table 319, Sequence No. 4 (Figure 4-4) states: "Add MH_EXP_TEN to WORF. ADJUST MH_RMN". This statement follows a prompt for, and entry of, MH_EXP_TEN from Sequence 3 of Table 316 (not shown). Sequence 4 of Table 319, also follows other actions (Sequences 2 and 3) that require data to be overlayed on the TPR. The Data element MH_RMN is contained in both the WORF and the TPR files. The first part of Sequence 4, table 319, is specific; "ADD MH_EXP_TEN TO WORF". The second part of sequence 4 ("ADJUST MH_RMN") is ambiguous, however, and the designer is not sure if "ADJUST MH_RMN" refers to the WORF only, TPR only, or to both files.

Figure 4-4 Decision Logic Table 319

Other typical problems found in the DLTs included the following:

- Prompt message omitted.
 - Error message not always provided for.
 - Data element name abbreviation not consistent with those provided in Annexes A, B, C, and D of the DFSR.
 - Incorrect placement of "X" marks in referencing sequences to Rules.
 - Inadequate information provided concerning contents of output messages, when not provided in Annex B.

4.2.2 R NET Contribution to Identification of DLT Deficiencies

R NET definition requires consideration of the availability of data as it flows through its processing steps and provides strong clues for identifying needed processing paths at nodes where branching decisions are made, particularly error paths that could arise due to the situation at these nodes. This R NET characteristic was a major factor in recognizing the logic errors and omissions found in the DLTs.

The R NET definition step is intolerant of ambiguity. In order to complete an R NET, all aspects of the data flow in the process must be known. As a result, several questions are always under consideration, such as:

- What GLOBAL DATA is needed?
- What LOCAL DATA is available from the MESSAGE which stimulated traversal of the R NET from outputs of preceding processing steps (ALPHAs), or from the initial values of LOCAL DATA which are set each time the R NET is used?
- What branching is appropriate, what DATA will be used for branching decisions, and what DATA values will determine which branch shall be traversed?
- What DATA is needed to FORM output MESSAGES that are to be transmitted as a result of R NET processing, what is the source of the DATA needed for the transformations necessary to MAKE the needed output MESSAGE, and what processing steps (ALPHAs) are necessary to accomplish the necessary transformations?

All ambiguous descriptions have to be directly addressed so that when the software engineer could not understand the intent as described, or when needed information to complete the R NET was missing, the problem quickly became apparent. Thus, the R NET development was the major contributor in the recognition of illogical processing, missing processing, and ambiguous descriptions of processing.

4.2.3 Special SREM Procedures to Assure Unambiguous DATA Naming

One of the most pervasive problems common to all software specifications is the difficulty in being specific about a named DATA item. The normal approach, which was also common in the MOM DFSR, is to try to use exactly the same name throughout. On the surface, this seems appropriate

but, in reality, that item can actually be several different data items during a process. Initially it may be in the input stream and be stored for later use, possibly in more than one GLOBAL file. Later it may be accessed and placed in still other GLOBAL files, or used for transmission in the output stream. These may be simply described as the transfer of the value resident from data of one type to another type. However, in describing the processing logic it is difficult to express the form of the data item being described without the use of considerable modifying information.

Take, for example, the DATA item UIC_CUST (Unit Identification Code Customer). On DLT 4, the item is used several times, and is referred to in several forms as follows:

Sequence No.	Reference to UIC_CUST
1,2,3	XMA UIC_CUST
1	UIC_CUST ON X_REF FILE
4	UIC_CUST ON WORF

Clearly, three different UIC_CUST DATA items are involved here. The writers of the MOM DFSR took pains to indicate which data item was intended, but at times the meaning of the descriptive modifiers was found to be ambiguous. So, even though the name: UIC_CUST will be found in Annex A and Annex B as a data item in input and output descriptions, and in multiple GLOBAL files of Annex D, it really is a different data item in each of those contexts. As we have seen, this is reflected by the need to use modifiers in the DLTs to indicate which UIC_CUST was being described.

The way this problem is handled within RSL on this effort was to use the basic name (UIC_CUST) and append a descriptive suffix. For example, if UIC_CUST MAKES an input MESSAGE, we used the name UIC_CUST_IN. Conversely, if it MAKES an output MESSAGE it was given the name UIC_CUST_OUT. In its GLOBAL form, ASSOCIATED WITH an ENTITY_CLASS or ENTITY_TYPE, it was given an appropriate suffix for that ENTITY. Some examples, showing the related ENTITY are:

- UIC_CUST_CRF_B
(CROSS_REFERENCE_FILE)
- UIC_CUST_XMX_DABS
(CROSS_REFERENCE_TRANSACTIONS_XMX_B_DABS)

- UIC_CUST_XME_A_DABS
(EQUIPMENT_RECALL_NEW_ITEM_XME_A_DABS)
- UIC_CUST_EORR
(EQUIPMENT_RECALL_REO)
- UIC_CUST_MPR
(MAINTENANCE_PROGRAM_REQUIREMENTS)
- UIC_CUST_UED
(USAGE_EXCEPTION_LIST_DATA_BASE)
- UIC_CUST_WORF
(WORK_ORDER_REGISTRATION_FILE)

By using this approach, we have unambiguously named all the forms that each basic data item can take. As a result, RADX can apply its various tests to each one of these DATA items and a more precise determination of consistency will result.

4.2.4 Identification of Consistency Problems via RSL and RADX

The major contribution of RSL is the discovery of consistency problems. For example, RSL requires that every element in the data base possess a unique name. Thus, if there is an attempt to name a MESSAGE with the name previously given to (say) a DATA item, a translation error code will be provided when an attempt is made to enter the MESSAGE into the data base. This prevents inadvertent duplicate naming of different elements.

The duplication naming problem just described is most useful when SREM is being utilized to create the software requirements from a system level requirement. Consistency problems in verification efforts are more often discovered during the RADX evaluation. But, in order to use RADX to discover these problems, a particular approach during R_NET development is required during the verification effort.

The approach used for verifying the MCM DFSR to prepare for RADX was to record the exact name of the DATA item, as given in the DLT, even if known to be wrong, and to use it as described for the processing, such as for branching decisions, or for ALPHA INPUTS or OUTPUTS, or to MAKE an output MESSAGE. Part of the RADX tests check DATA consistency by establishing the SET of DATA that is produced by the DP system (source DATA) and a SET of DATA that is used by it (sink DATA) and then comparing the two SETs for mismatches. DATA is considered to be source DATA when it:

- MAKES an input MESSAGE.
- Is OUTPUT FROM an ALPHA.
- Is LOCAL but has an INITIAL_VALUE.
- Is CONTAINED in a FILE which:
 - MAKES an input MESSAGE.
 - Is OUTPUT FROM an ALPHA.

Data is considered to be sink DATA when it:

- MAKES an output MESSAGE.
- Is INPUT TO an ALPHA.
- Is used (REFERRED) by the R_NET for branching decisions.
- Is RECORDED BY a VALIDATION_POINT.
- DELAYS an EVENT on the R_NET.
- ORDERS a FILE (establishes a desired order of DATA instances).
- Is CONTAINED IN A FILE which:
 - Makes an output MESSAGE.
 - Is INPUT TO an ALPHA.
 - Is RECORDED BY a VALIDATION_POINT.

All DATA that are produced (source DATA) should also be used (sink DATA) and, conversely, all that are used should be produced. A portion of RADX is designed to make that check. DATA may be found via RADX that has neither sink nor source. When this occurs, it is DATA which has been named but never given a relationship with any other element such that it could qualify as either sink or source DATA. DATA which has a source but not a sink, or a sink without a source, usually results from not being consistently named in the specification being verified.

If a mismatch between source and sink DATA occurs, it does not automatically follow that the mismatch is the result of inconsistent DATA naming. It is also possible that some other problem may exist. Some examples:

- An output MESSAGE was FORMED BY an ALPHA using DATA produced by the ALPHA, but the MESSAGE was not actually defined separately as being MADE BY the DATA in question.
- An input MESSAGE was defined and the DATA that MAKES the MESSAGE indicated. Inadvertently, that MESSAGE was never defined as being FORMED BY an ALPHA on an R_NET.
- An EVENT was defined as being DELAYED BY a DATA item (which actually is a predefined constant indicating the length of the delay), but the DATA item was not given an INITIAL_VALUE.

Thus, it can be seen that when there is a mismatch of source and sink DATA, the software engineer has to investigate the reason.

4.2.5 Identification of Consistency Problems by Observation

When the approach given above is used, inconsistent DATA becomes imbedded in the data base. For a normal verification effort this is acceptable, and the inconsistency is removed only when the developer actually corrects the inconsistent name in his specification. In our efforts under this contract, however, we were required to demonstrate the REVS tools and their use for Data Flow analysis and regeneration of the requirements. With imbedded problems, neither of these demonstrations can be properly accomplished. Therefore, only a few inconsistencies were allowed in the data base to illustrate the RADX capabilities to discover them. For the most part, however, inconsistencies recognized by the software engineers during this effort were recorded in Trouble Reports, but corrected before being entered into the data base so as to better support the Data Flow Analysis, and to allow an example of regenerated requirements to be produced via the RADX documentation capability.

4.2.6 Identification of Problems by RADX

The primary role of RADX is the determination of inconsistency and incompleteness of the information in the data base. Nearly all the sets are created for those purposes. The results of the RADX runs which were accomplished at the completion of the data base, are provided in Annex C. How RADX was used was the subject of earlier discussions in Section 2 and 3 and, therefore, will not be repeated here. The results of the findings of the RADX runs shown in Appendix C are summarized in Paragraph 4.4.

4.2.7 Summary of Deficiency Findings

A total of 302 Trouble Reports have been written as a result of the verification effort on the MOM DFSR. A percentage breakout of the kinds of deficiencies reported is shown in Figure 4-5.

The largest category of deficiencies was inconsistency. This is not surprising, given the large number of data names that were involved and the amount of multiple data naming caused by the format of the specification. The process of coordinating the data naming must have been a large job since many people apparently worked to produce the document. This factor alone provided significant opportunities for inconsistent naming. The benefit of the automated REVS tools is apparent when compared to the manual approach undoubtedly used in the MOM DFSR. Suppose it is discovered that a data item inadvertently had been given two names and that an effort must be mounted to correct this inconsistency. In the manual mode, many hundreds of pages would have to be inspected to assure that the incorrect name was changed to the one that was to survive. It would probably take an hour or two to check through all the pages of the DFSR in such an effort, plus the time necessary to retype all the affected pages. Compare that with how we would fix the problem with RSL. Suppose the two DATA items were in the data base named AAA and AAB, but the correct name was AAA. We would simply enter the RSL command:

MERGE AAB into AAA.

As a result of this command, all the relationships and all the attributes of AAB would be assigned to AAA, and the DATA item AAB would then be purged. Thus, if relationships and attributes equivalent to those used in Annexes A, B, C, and D were established by extension of RSL (as was done for the requirements regeneration example), this one command would correct the inconsistency everywhere it existed and we could be assured that it had been. How long would it take? Perhaps 15 seconds at the keyboard and a very short computer run.

Deficiencies in the categories "Ambiguous" and "Missing" were the next most prevalent types, followed by "Illogical" and "Incomplete", in that order. We have further segregated the reasons behind these categories of deficiencies, and they are shown in Figures 4-6 through 4-10, along with some limited Trouble Report examples of the Category of Problem being displayed. The length of each bar on these figures is related to the

percentage of all the Trouble Reports (left hand scale) in the indicated CATEGORY_OF_PROBLEM. The number at the end of each bar indicates the actual quantity of Trouble Reports involved. The full listing of all Trouble Reports can be found in Appendix D.

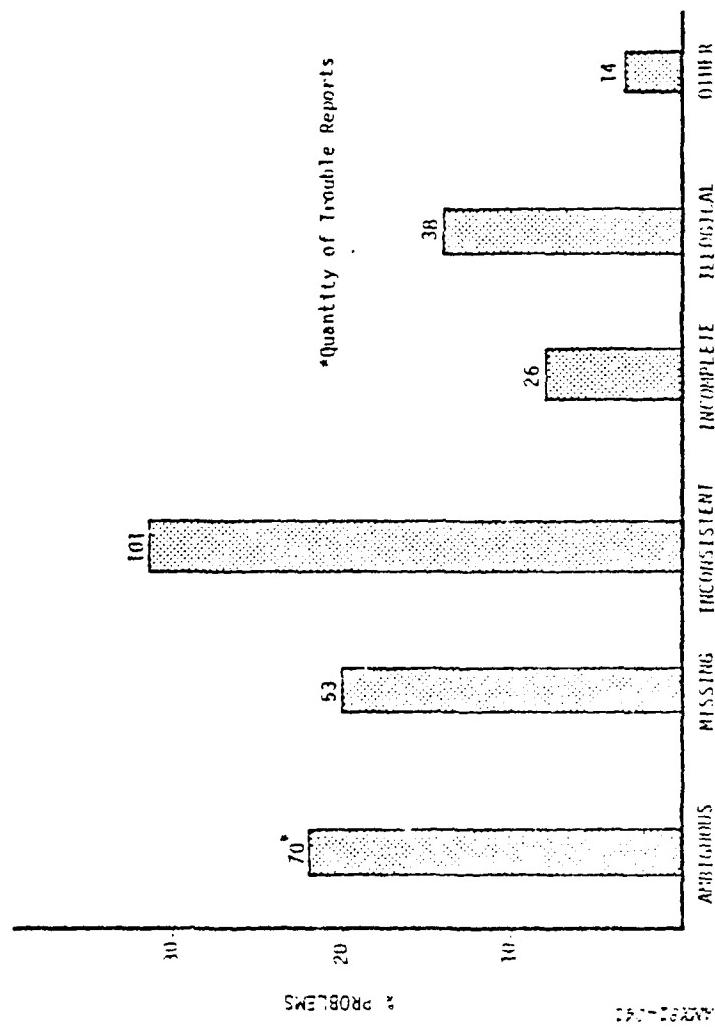
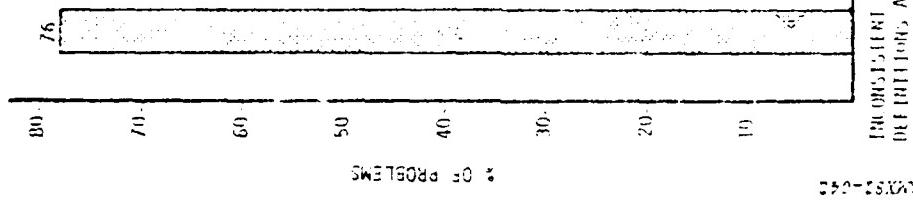


Figure 4-5 Distribution of MOM DFSR Deficiencies



Typical Examples

- MOD 9 AND MOD 17 * Decision logic tables infer one definition for a data item. Annex D states a different definition for the same data item.
- Rule 25 Two different references in the specification give the same data item different field lengths.
- MOD 35 (UT #1601) states that the SUPPLY STATUS and SUPPLIER STATUS is to be merged and sorted by DOCU NO. DOCU NO does not exist within SUPPLY STATUS nor SUPPLIER STATUS.
- MOD 45 Decision logic tables require that a validity check be made on INPUT NO fld input against IDFLN NO 11 on IPR. Data item IDFLN NO fld is not included in the Input element for the IPR.

* Trouble Report Number

Figure 4-6 Components of CATEGORY OF PROBLEM: Inconsistent

TYPIICAL EXAMPLES

- **MOM 80*DT 1674.** The processing to be done here is unclear. Items to be compared are not specified. A part number comparison between input XMP information and the Bench Stock File has been assumed.
- **MOM 101 DLT 1702** uses the term Overhead Record. The meaning of Overhead Record cannot be determined and intended processing is uncertain.
- **MOM 106 DT 1703** shows the data item TMRSCIN DATE ORD being used. This Data Item is not contained in the PRK file. DATA PRTP_ORD is assumed to be the proper Data Item for this processing.
- **MOM 84 DT 1675** does not specify what changes are to be posted to the PRK. Also there is no data item HDO_CDR_R0 input for this processing, although it is shown as being used.

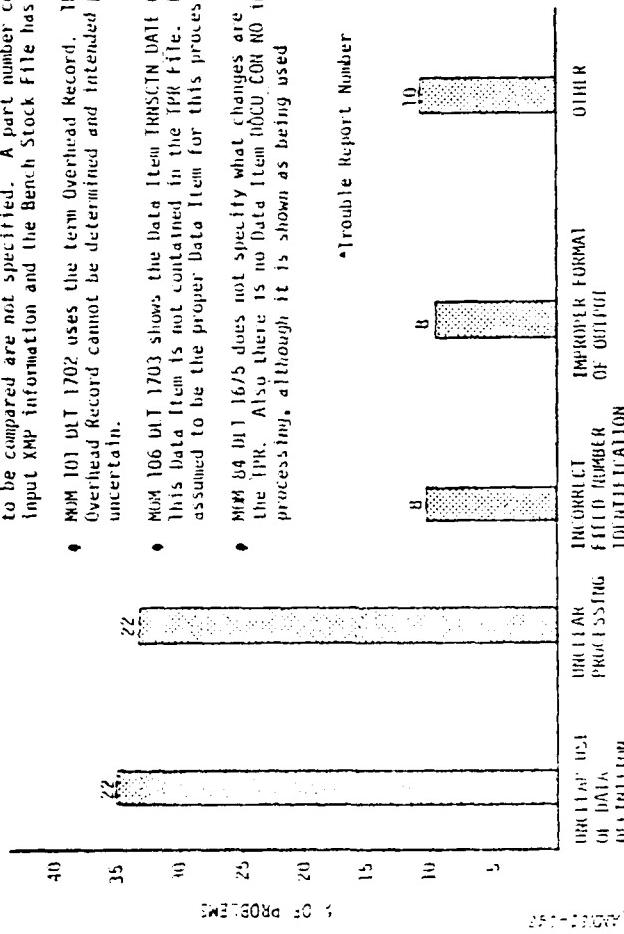
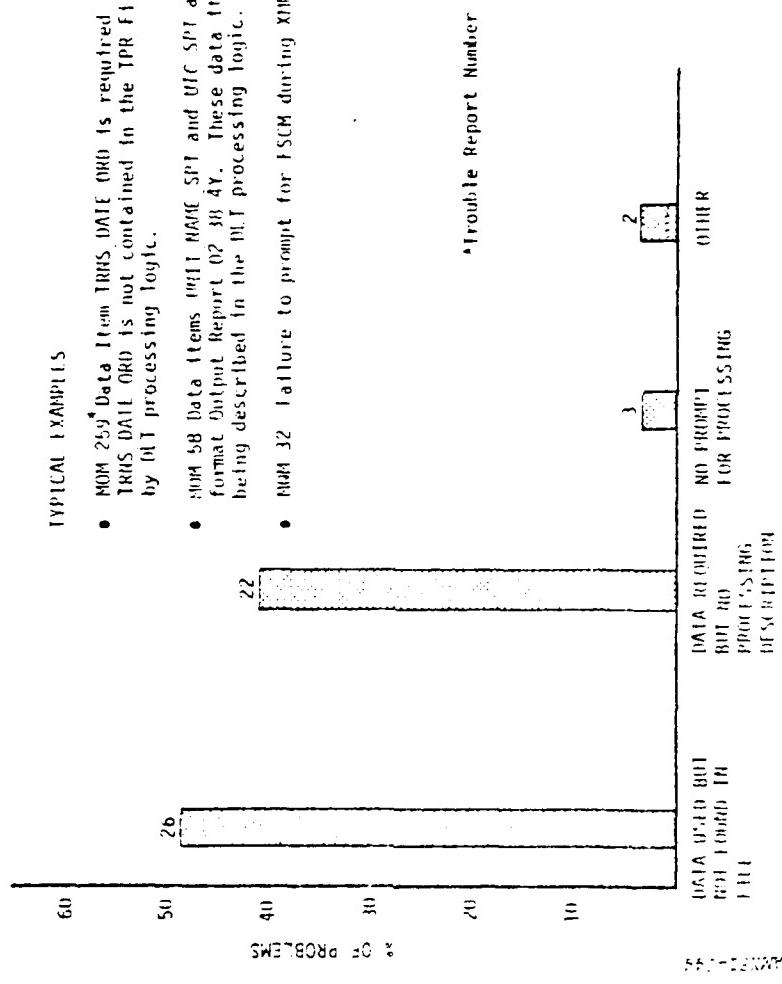


Figure 4-7 Components of the CATEGORY OF PROBLEM: Ambiguous

TYPICAL EXAMPLES

- Item 259 Data Item IRMS DATE (RD) is required for Output Report 02 37 AW.
This DATE (RD) is not contained in the TPR file and is not furnished by MLI processing logic.
- Item 58 Data Items MLI NAME SPT and UIC SPT are required to format Output Report 02 30 AW. These data items are used without being described in the MLI processing logic.
- Item 32 Failure to prompt for FSCM during XMB.



三

10

11

三

Typical Examples

- NMN 254 * Sequence #1 shows DLT 2806 as stating a requirement for entry to this DLT. There is no DLT 2806. It is assumed that the proper entry to this DLT is DLT 2863.
 - NMN 255 DLT 2864 shows DLT 2866 as stating a requirement for entry to this DLT. DLT 2866 does not contain a reference to this DLT.
 - NMN 14 DLT 111 Sequences 1 and 2 provide for validity checks of USE MRK R(F) HR to determine if the data has an alphanumeric value or is blank. The rules, as now written, require the entry to contain an alphanumeric value and to be blank at the same time.
 - NMN 153 DLT 2103. Processing specified for Sequence 4, 5, and 8 for data item UIC CUSI on Sequence 6 has already been accomplished by DLT 2101

Figure 4-9 Components of the CATEGORY OF PROBLEM; Illogical

70

Typical Examples

- MAM 220M 31 checks for proper prior letter for a previously entered MAM Shift 01, but no check is made to determine if the current one has been previously used.
- MAM 107 (11) 101 requires the printing of the output report 02 85 4M th its entirety. Only parts I, II, III are formatted within the 1700 series of M1's, and parts IV and V are not.
- MAM 95 page H147 states that Part V of Output 02 35 4D will be completed for Receipts with XMK CRG DS6 A. Although considerations of XMK receipts are treated beginning with Part 1b4D, Part V of Output 02 35 4D is neither formatted nor printed in this series of M1's, as would be expected.

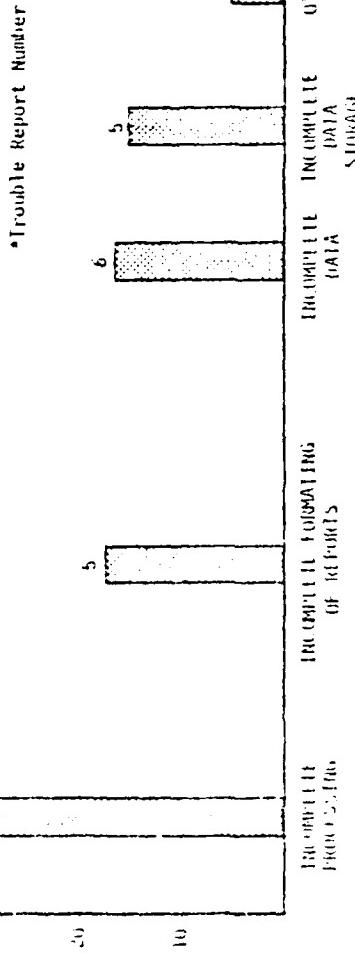


Figure 4-10 Components of the CATEGORY OF PROBLEM: Incomplete

4.3 MAJOR MOM DFSR PROBLEMS

Previous paragraphs have outlined the kinds of deficiencies that have been reported via Trouble Reports. As in all verification efforts we have accomplished using SREM, certain problems are identifiable as possessing higher criticality because they represent situations which typically result in serious difficulties for the software designer in implementing the intended processing described in the requirements. They may be individual omissions or ambiguities of significant import. Or they may be groups of problems which individually are minor in nature, but the quantity of which are significant enough to elevate the group to the higher level of criticality described in this paragraph.

It is important to note that the source of these deficiencies is solely the Decision Logic Tables (DLTs) in Annex H of the DFSR, as supported by Annexes A, B, C and D for data definition. As described in an earlier portion of this report, we concluded that the DLTs were the most complete DFSR description of intended processing, and as a result, our efforts were focused on them for the verification effort. In so doing, we concluded that all appropriate processing requirements should be covered in the DLTs so that the software designer could expect to find all the information he needed in these tables. Thus, he shouldn't have to search for obscure footnotes, or through a large quantity of text, for the key information he needs to develop his design.

This approach may seem restrictive in the context of this DFSR's since it is rich in information, and since significant effort has been expended in attempting to completely portray not only the necessary processing, but also the context of the overall operations within which the processing requirements reside. However, good requirements writing practice suggests that fewer errors result when the software designer finds all the information he needs in expected, contextually-appropriate locations within the requirements document. Consequently, the discussion that follows should be read in the light of these comments. Said another way, some of the deficiencies outlined in following paragraphs actually are addressed in portions of the DFSR other than the DLTs, but our view is that the requirements details should be consistent and complete within the DLTs, since that is the portion of the DFSR on which the software designer is

likely to rely for determining the processing requirements his design is expected to satisfy.

The areas of the requirements which we feel could cause significant problems for the software designer, and which are individually discussed in following paragraphs, are as follows:

- Failure to initiate certain batch processing output reports.
- Failure to use the parameter report controls.
- DABS production and use inconsistencies.
- Work Order Number character field confusion.
- Lack of file purge instructions.
- Missing file contents.
- Significant quantities of individually minor deficiencies:
 - Inconsistent data naming.
 - Incorrect DLT referencing.

4.3.1 Failure to Initiate Certain Batch Processing Output Reports

Paragraphs 4.11 and 5.9 indicates that the operator will key for the accomplishment of cyclic (daily, weekly, and monthly) batch processing efforts to produce periodic output reports. The DLTs which produce these reports are not referenced (called) by any other DLTs because the initial input (keying by the operator) is not found in a DLT to start the batch processing that leads to the output of the periodic reports, nor is an input description provided in Annex A to indicate that an input message is necessary to initiate this processing.

Certain other output reports are defined in DLTs which would be expected to occur during the chain of periodic preparation of output reports. However, these reports would not have been referenced even if there had been a DLT showing the operator keying to initiate periodic report processing. Whereas all the other periodic reports are referenced from one to the next (i.e., initiation of a DLT series is called from the completion DLT of a preceding series), the initiation DLTs of three reports are not referenced from any other DLT.

A third category of output report deficiencies is those cases where one or more parts of multi-part output reports are not called for processing by a DLT, even though the other parts are. Three different output reports fall into this category.

A fourth category of problem exists in the DLTs where five output reports were not addressed by any DLT. That is, the description of how those output reports were to be developed and formatted was not defined on any DLT. Table 4.1 lists the periodic output reports whose problems are described above.

4.3.2 Failure to Use the Parameter Report Controls

The parameter for report control is added to the header segment of several files via entry XMZ (Card Designator Code SAMS: E), and this input is described in DLTs. However, these parameters are never used in forming the daily, weekly, and monthly output reports, as would be expected in light of the comments of page A-206 which describes the use of these parameters.

4.3.3 DABS Production and Use Inconsistencies

The Daily Accumulated Batch Storage (DABS) is used as a temporary hold file for daily inputs entered into the system. The information describing the various input formats to be saved by DABS is found in Annex D (Page D-20). The "Remarks" block of Page D-20 lists the input to be saved. Five of the inputs described in the DLTs as being processed are not indicated as required on Page D-20. Conversely, the DLTs do not show the logic necessary to process DABS storage of nine files that are required by Annex D.

To complicate the problem, the input descriptions of Annex A also indicate which inputs are to be written to DABS. These are inconsistent with the similar information indicated in Annex D. In addition, five of the inputs shown in Annex A as being stored in DABS are not defined in any DLT as being so stored. Conversely, the DLTs indicate storage to DABS of five inputs not marked for storage in Annex A. Table 4.2 lists these DABS inconsistencies.

Table 4.1 Periodic Output Message Problem

Output Report Number	Print Number	Output Report Name	Portion Not Formatted
02 32 40	1673	PARTS AMALTING DISTRIBUTION ACTION	PARTS 1 & 2 OF 3
02 33 41	1676	SS/WORK ORDER ISSUE CANCELLATION LIST	ALL
02 35 40	1670 1671	DAILY SUPPLY TRANSACTIONS	PARTS 2 & 3 OF 5
02 42 88		LABOR UTILIZATION DETAIL	ALL
02 61 88		MESSAGE EXCEPTION LIST	ALL
02 80 88		WORK ORDER DATA	ALL
02 83 80	1671	SUPPLY ACTIVITY REQUIREMENTS FORM/UP DOCUMENT NUMBER	ALL
02 83 80	1670	SUPPLY ACTIVITY REQUIREMENTS CANCELLATION REQUEST FORM/UP	ALL
02 86 10	1671	SUPPLY ACTIVITY REQUIREMENTS	PART 1 OF 3
02 87 88		SUPPLY RECONCILIATION RESPONSE	ALL
02 90 10		SUPPLY RECONCILIATION RESPONSE	ALL

Table 4.2 Daily Accumulated Batch Storage (DABS) Inconsistencies

DABS INPUT PROCESS REQUIREMENT	STORAGE TO DABS SHOWN IN:			REMARKS
	ANNEX A	ANNEX C	ANNEX H	
(MD)	•	•	•	•
(ME (A))	•	•	•	•
(ME (B))	•	•	•	•
(MF)	•	•	•	
(ML)	•	•	•	
(MM)		•	•	
(MN)		•	•	
(MP (A))		•	•	•
(MP (B))		•	•	•
(MP (C))			•	•
(MP (D))	•		•	•
(MP (E))	•		•	•
(MP (F))	•		•	•
(MP (G))	•		•	•
(MR (A))	•	•		•
(MR (B))	•	•		•
(MT)		•	•	
(MU)	•	•	•	
(MV)	•	•	•	
(MX (A))	•	•		•
(MX (B))	•	•		•
(MZ (A))		•		•
(MZ (B))		•		•
(MZ (C))		•		•
(MZ (D))		•		•
(MZ (E))	•	•	•	•

AMSL-077

• (MD PROCESS WRITTEN TO DABS ONLY WHEN VALUE OF TASK_PARTING_ID IS '1').

↔ LETTERS IN PARENTHESIS INDICATES VALUE OF CARD_DSG_ID_DAMS.

4.3.4 Work Order Number Character Field Confusion

After the MOM DFSR had been published, one of several changes involved the deletion of a data element: P_WON (Partial Work Order Number). It was to be understood that the data element: "Work Order Number" was to supplant P_WCN wherever it appeared in the DLTs, but the actual changes to each affected DLT page were not accomplished.

Unfortunately, this change was not straightforward. Because P_WCN had nine characters while the Work Order Number had 12, problems developed where subportions of P_WON were cited on DLTs. The subportions of P_WON and the Work Order Number (as to character location) are not consistent. Because of the pervasive use of P_WON throughout the DLTs, handling the change in this fashion probably assures that subsequent designers, who may not have been familiar with P_WON, will be confused with the DLTs as now written. As much trouble as it might be, there would be a distinct benefit

in modifying all DLT references to P_WON to those appropriate for the Work Order Number.

4.3.5 Lack of File Purge Instructions

The summary sheets for various files in Annex D indicate, in one way or another, that the files are to be purged. However, with the exception of the WORF and TPR, no specific indications are provided in the DLTs to describe under what processing conditions a specific file is to be purged. Failure to provide this information may result in unintended results if the designer misinterprets the variety of notes in various places concerning purging (none of which precisely define the purge conditions).

Thus, if the purge decision is to be made under software control, this control should be defined in the DLTs. If the purge is under operator control, an appropriate input message (or messages) should be defined and a DLT for each should define how the purge is to be accomplished and what protective considerations are to be included.

4.3.6 Missing File Contents

The Shop Stock and Requisitioning Process (Shop/Shipment Status (AE_, AS_, PU_, XMR(B)) Daily Update Subprocess contained in DLTs 1601 through 1630 require the Supply Status File and Shipment Status File as the basis for Output Reports 02-35-4D, 02-34-4Y, and also parts of 02-99-4R. These two files are not defined by Annex D (File Descriptions!).

For the purpose of R-NET definition, we assumed these files were made up of the data items contained in Annex A, pages A-66 and A-71. However, to assure that such assumptions do not have to be made by the software designer, these files should be defined fully in Annex D.

4.3.7 Significant Quantities of Individually Minor Deficiencies

Two observed areas of deficiency are singled out as important problems by virtue of the quantity of deficiencies documented. These are 1) Inconsistent data naming and 2) Incorrect referencing of DLTs.

Inconsistent data naming is a problem found in virtually all software requirements specifications. In spite of obvious attempts to attain consistency, the sheer quantity of data items involved in a system of this type just about guarantees the introduction of errors when a manual system

of requirements definition is attempted. This problem was present in the MOM DFSR, particularly in the area of batch processing. Unless corrected, unintended introductions of spurious data may occur. As a result, one designer or coder may use one data name, but the different names may be inadvertently used by others, thus causing considerable later problems in finding and correcting the problem.

The second pervasive problem area is that of DLT referencing. We believe the DLT approach is one of the better ways of describing intended processing that we have seen. They clearly show the kinds of decisions intended and the order of processing required. The problem we experienced stemmed from inaccurate referencing from one DLT to the next. Referencing was sometimes to the incorrect DLT, and sometimes to non-existent DLTs. Problems of this kind often arise when an initial set of well referenced DLTs is subjected to a change (insertion of new DLTs, or deletion of existing ones), and subsequent consistency checking of referencing among the new set of DLTs is not completely accomplished. As mundane as this kind of error is, it must be realized that many more hours of software designer efforts will be required to figure out what was intended, than will be expended by the requirements engineer to modify the DLTs to show proper referencing.

4.4 FINDINGS OF SREM PHASE RADX RUNS

A normal step in the SREM process involves application of a standard set of static RADX checks to the data base to identify problems introduced during its development. RADX efforts under this contract were constrained by available Government furnished data processing time which was limited to 4 hours. In preparation for Regeneration of Requirements we extended RSL to contain appropriate elements, attributes, and relationships to duplicate several of the annexes in the current DFSR directly from the data base. Although this approach is described in more detail in paragraph 4.5, it is mentioned here since the larger data base required more processing time than is normally experienced in a typical SREM application. As a result of the processing time limitation, it was not possible to completely apply the RADX tool, nor to completely regenerate the requirements specification. Instead, we have developed examples of these processes to illustrate SREM's capabilities in these areas.

A portion of the standard set of RADX checks for Phase 1 and 2 of SREM was applied to the data base. In addition, Data Flow Analysis was applied to a portion (one input message) of the data base. These results are illustrated and discussed in Appendix C.

4.5 REGENERATION OF REQUIREMENTS

With the completion of the requirements data base, a wide variety of documentation is possible. This variety stems from the RSL/REVS capability to:

- Control the items to be listed by the establishment of SETS of HIERARCHIES of interest.
- Control the amount and type of information to be displayed for each SET or HIERARCHY of interest by the use of the APPEND statement.
- Define and document any element of interest with its appropriate relationship to other elements, and with its attributes through RSL extension.

Our approach was to illustrate how portions of current DFSR documentation might be produced directly from the data base. Although literal copies of tables can not be directly produced from the data base, all the information can be developed, entered into the data base, and these produced in various ways to represent the DFSR documentation. Thus, information was developed to document the following DFSR documentation:

- Annex A - Input Descriptions
- Annex B - Output Descriptions
- Annex C - Data Element Descriptions
- Annex D - File Descriptions
- Annex E - Decision Logic Tables.

In addition, other documentation can be produced, such as a totally cross-referenced listing of the entire data base.

This approach represents a thorough, yet succinct, description of the MOM DFSR processing required, and of the data elements involved in the processing. The most important aspect added through the use of the requirements data base to produce this documentation is consistent data naming and the retention of consistency when data element names are changed. For example, if a data element name must be changed, a simple input is made to the data base which guarantees that this data element name will be consistently changed every place the previous name appeared in any of the documentation.

An even bigger benefit of documentation under the SREM approach will be realized when there are changes to the processing requirement. No matter whether the change is an addition, an insert, a deletion, or a change to that described in the data base, the RADX checks provide assurance that the modification has not introduced unintended problems elsewhere in the requirements. Or if it has, the problems that result are clearly presented for corrective action.

The reader is invited to review the documentation examples provided in Appendix B. There, each example is described and illustrated to demonstrate the capability to provide consistent information to document the MOM DFSR as a result of this SREM application.

5.0 A SYSTEMS ENGINEERING APPROACH TO THE EVALUATION OF REQUIREMENTS METHODOLOGIES

One of the thorniest problems in requirements development is evaluating the impact of different requirements methodologies on the software development process. Such an evaluation technique is mandatory for a systematic comparison, and for selection of a software requirements methodology which is most effectively applied to a specific project. In this section, we will present a systems engineering approach to methodology definition and evaluation. This is accomplished by imbedding the requirements generation in an overall life cycle development context and defining the inputs, outputs, and performance indices of the requirements methodology. It is motivated by the observation that the purpose of a methodology is to output a specified set of information, such as software requirements, in a sequence of logical steps to produce the final product; in this case, the tested software product.

Our discussion will compare the following requirements techniques to SREM:

- The Jackson method.
- CADSAT (or other PSL/PSA versions).
- HOS.
- SADT.
- IORL.

We will first briefly describe and compare these techniques to SREM. Following that, we will discuss the importance of comparing the cost and performance of such techniques in the context of the life cycle cost and resulting performance of the software system. Finally, we will assess and compare the life cycle cost of each technique.

5.1 TECHNICAL COMPARISON OF SREM TO OTHER TECHNIQUES

It is impossible in a few short pages to adequately describe the many requirements and specification techniques and compare and contrast them with SREM. In spite of this, comparison of a few relevant techniques is worthwhile to highlight some unique features of SREM. For this purpose, a few of the more important techniques, as previously listed, were selected for comparison. The technical comparisons described below are summarized in Table 5.1.

Table 5.1 A Comparison of Some Requirements Techniques

	STW/REVS	JACKSON	PSL/PSA	HOS	SADT	ICRL
<u>CONTENT</u>						
PROCESSING FUNCTIONS DATA I/O	E	E	E	E	E	E
SEQUENCES OF FUNCTIONS FOR ONE INPUT	E			E		E
SEQUENCE OF I/O	P	E		E		E
FUNCTION HIERARCHY (STRUCTURE)	P	E	E	E	E	E
TRACEABILITY	E		P			
PERFORMANCE REQUIREMENTS	E					P
<u>AUTOMATED TOOLS</u>						
- LANGUAGE	E	M	E	P	O	E
- CONSISTENCY CHECKING	E	M	P	M	O	E
- DATA FLOW ANALYSIS	P	M				
- SIMULATION GENERATION	E		P			
- USER EXTENSIBLE	E					

E EXPLICITLY ADDRESSED
P PARTIALLY ADDRESSED
M MANUAL
O IN DEVELOPMENT

PRBS0-731

All of the compared techniques define processing in terms of "functions" with inputs and outputs. It is interesting to note that SREM, HOS, and ICRL attempt to define a stimulus/response relationship of inputs to outputs, while Jackson, PSL/PSA, and SADT express data flow but not

precedence or control flow. All techniques, except SREM, explicitly define processing in terms of a "hierarchy of functions", whereas SREM is based on a "flat" graph model which can be expressed in terms of a hierarchy of subnets -- a subtle but important difference.

Sequences of inputs and outputs are explicitly defined by Jackson's technique, partially defined by SREM, but not defined by other techniques. The comparison of SREM with Jackson's technique is interesting: Jackson emphasizes definition of a life cycle of inputs about an object, and describes the life cycle of processing those sequences, and thus derives the information (state) which must be kept in the data base about the object. SREM requires the identification of ENTITY_CLASSES (objects about which data is maintained in the data processor), and the ENTITY_TYPES which compose them (states of the entity which require maintenance of unique subsets of data), and the DATA which is ASSOCIATED with the ENTITY_CLASS and ENTITY_TYPES. Input and output messages are identified with the ENTITY_TYPE, stimulus/response requirements are expressed in terms of graphs of functions, and these are then merged together to form the R_NETs. Thus, the sequences of I/O messages are partially defined by the transitions between ENTITY_TYPES.

SREM explicitly provides for the expression and analysis of traceability between a set of ORIGINATING_REQUIREMENTS and the final processing requirements. Versions of PSL/PSA have also incorporated this capability. SREM is the only technique which addresses the explicit definition of performance requirements (response times and accuracy of the processing from input to output). This is done in four steps:

- (1) Paths of processing are specified in terms of VALIDATION_POINTS on the R_NETs.
- (2) The paths are matched with the ORIGINATING_REQUIREMENTS which are applicable.
- (3) The ORIGINATING_REQUIREMENTS performance numbers are decomposed and allocated to the paths of processing in a series of tradeoff studies.
- (4) A PERFORMANCE REQUIREMENT is defined which CONSTRAINS the path, and is given as an attribute TEST which inputs specific data from the validation points, and outputs either "PASS" or "FAIL".

The result is a set of PERFORMANCE_REQUIREMENTS with pre-conditions and decision points on the R_NET (input data is valid but with an out-of-range measurement), functional post-conditions (specific data accessed, specific data updated, specific message output), and performance post-conditions (response time and testable I/O accuracy requirement). The R_NET thus provides the mechanism for graphically presenting similarities and differences of conditions for path expressions.

When we compare automated tools, we find that SREM has an automated language RSL with tools to check static DATA consistency, the dynamic DATA consistency processing of a single MESSAGE, limited consistency checking for sequences of MESSAGES (DATA is initialized when a new ENTITY_TYPE is set) and tools to support simulation generation. The language and report generation facilities are truly user extensible, allowing a user to add new elements, attributes, and relationships, input instances of the new elements and relationships, and retrieve them on the same run.

Both Jackson and HOS techniques are (at the time of this writing) currently manual, but tools are being developed. SADT has been partially automated by Boeing Computer Services as AUTOIDEFO. IORL has automated tools on a mini-computer for defining information in a text-file and diagram data base, and limited consistency analysis is available via comparison of parameter tables.

The comparison of RSL to PSL is worth special attention. PSL defines processing requirements in terms of elements, attributes, and relationships by defining a hierarchy of functions, data, etc. R_Nets were defined as a mechanism for defining sequences of processing requirements before RSL was defined. In seeking an approach for defining a language expressing these concepts, TRW noted the PSL work and decided to express RSL in terms of elements, attributes, relationships, and structures (R_Nets, SUBNETS) to define the stimulus/response conditions. REVS was then developed using the FORTRAN Data Base Management System used by PSA. In turn, later versions of PSA incorporated data consistency checking techniques first developed for REVS, and techniques for automated simulator generation (first developed for REVS) are under development for PSA. Thus PSL/PSA and RSL/REVS have had a substantial interactive effect on one another.

The combination of a precise machine processable language with which to describe the software requirement consistently and unambiguously, an

integrated consideration of structures which define the logic of processing in terms of the data flow, the automated capability to test consistency, completeness and traceability of the requirements data base, the automated capability to build, exercise, and analyze a simulation of processing using the actual requirements data base, all accomplished in precise ways established within the methodology, makes SREM a truly powerful software engineering tool. Other competing software engineering tools have some comparable features to portions of SREM's integrated capability. But none of them possess all the features of SREM. TRW insisted from the start that SREM be developed in accordance with carefully thought out, formal foundations, and that it be capable of supporting the complete software requirements engineering task.

Another reason why SREM excels is due to its approach of defining logic of processing in an R_NET as a response to each of the input messages that the data processor may receive. The competing software engineering tools still maintain the functional hierarchy as a starting point. We believe that if functions are to be defined, they can be more appropriately defined after the completion of the definition of processing logic which occurs from the stimulus-response approach. At that time, appropriate partitioning becomes more obvious and results in less arbitrarily defined functions.

Two other factors suggested the appropriateness of the stimulus-response approach, as the methodology was initially being designed. The first factor we observed was that, with requirements defined under the traditional functional-hierarchy approach, the first thing testers had to do was identify the various paths of processing in the system so that they could determine what software testing was appropriate. We decided that if this was to be done anyway, why not write requirements that way to start with? The second factor we observed was that preliminary or process design also depends heavily on understanding the logic of processing flow. Here again, it made better sense to provide the designer with the headstart that the stimulus-response approach provides.

This completes our look at technical comparisons. Let us next turn to another way to evaluate requirements techniques; a look at the life cycle costs for using them to develop and maintain software requirements.

5.2 THE SYSTEM ENGINEERING APPROACH FOR EVALUATION

The starting point of this discussion is presented in Figure 5-1, which illustrates that the life cycle methodology will input a problem statement and all subsequent modifications, and will output a sequence of versions of the software documentation and product; the methodology ends when the last version is retired from service. Note that the output includes not only the software product, but all levels of documentation and development effort for the product. This includes documentation of the requirements, their allocation to design elements, interface design, test plan, and the development plan (including cost and schedule) for component development, integration, and test (including the development of any necessary integration of test tools and test procedures). The performance criteria appropriate to the overall methodology is the total life cycle cost and estimates of the performance of the software system produced by each methodology. Note that these are separate indices so that tradeoffs may occur between them.

The life cycle cost can be expressed in terms of the costs of developing each version, plus the costs of operating each version. The system performance is a function of the operating performance of each version of the product. The cost of the methodology to develop a revised version of the product is dependent on the set of tools output by the last version, and the lack of such tools would increase the costs of developing the revised version substantially.

The performance indices of the activities to plan and to coordinate design and development can be stated as the overall cost and schedule of the development effort. However, this does not allow an assessment of the true total cost of requirements. That is, it does not show the downstream cost of fixing requirements errors, nor the costs of modifying the requirements and design in response to changes in the original requirements. To get to this level of detail, we must decompose the methodology to highlight these activities.

Figure 5-2 highlights the activities of fixing errors and modifying the requirements and design. The equations at the bottom of Figure 5-2 present relationships between the overall cost to develop the first version and the performance indices of the activities in this figure. A necessary consideration of the activity to define requirements is not only the cost

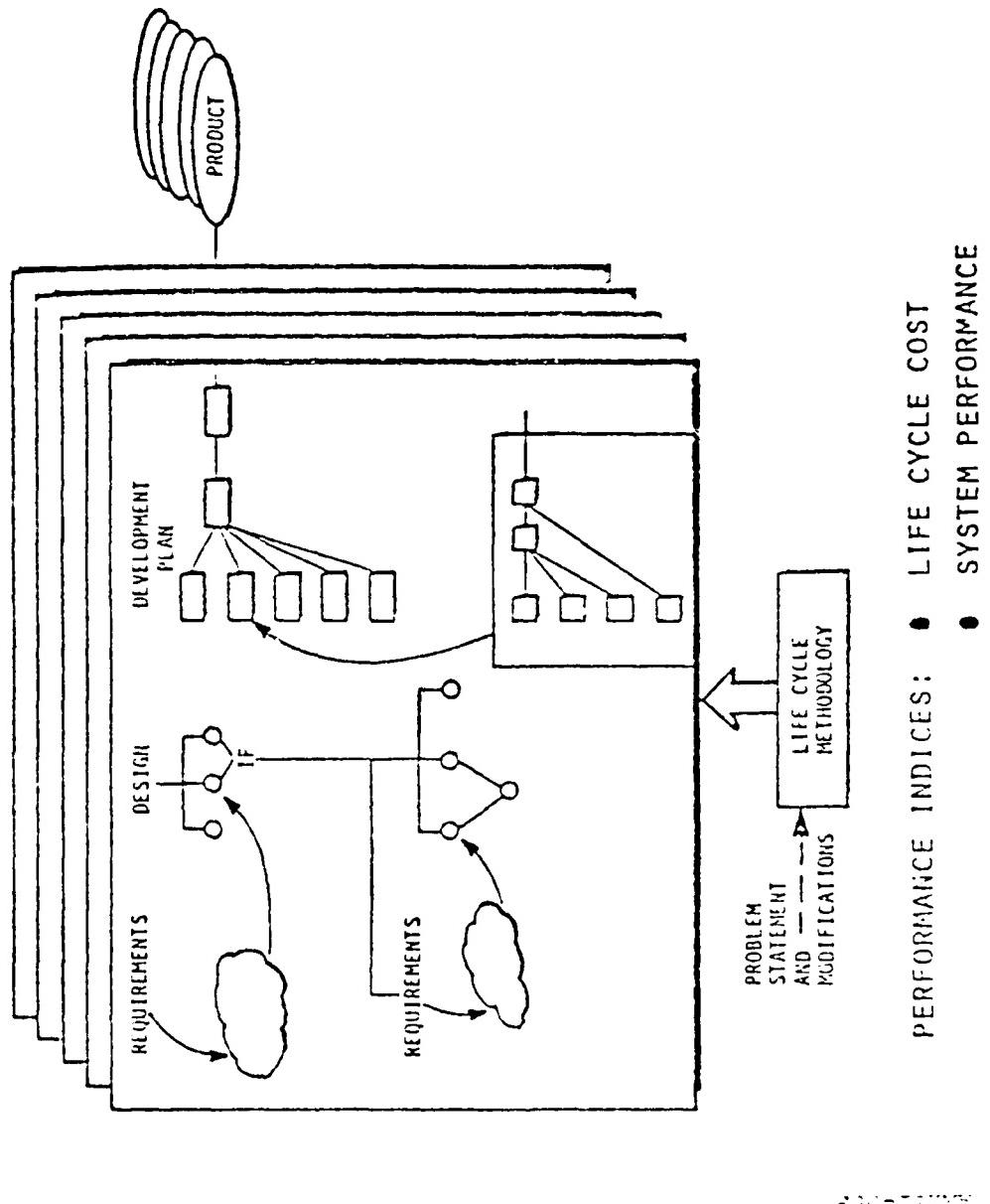
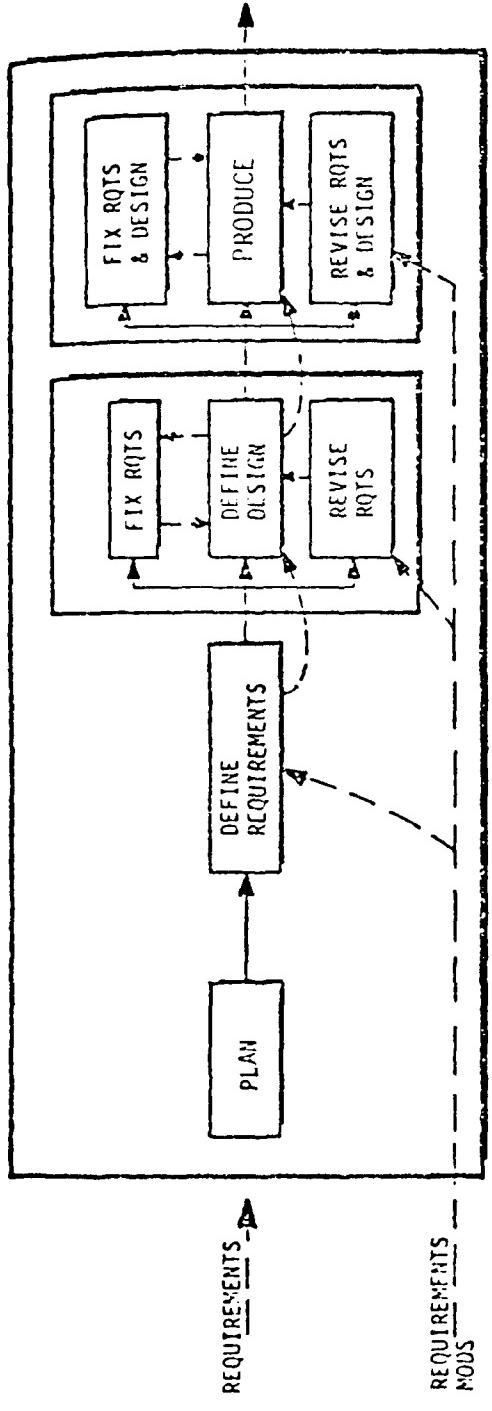


Figure 5-1 Methodology Evaluation Starting Point



PERFORMANCE INDEX: NR_REQUIREMENTS_ERRORS

PERFORMANCE RELATIONSHIP:

$$\text{COST_DEVELOP} = \text{COST_PLAN} + \text{COST_REQUIREMENTS}$$

$$+ \left[\text{COST_DESIGN} + \text{COST_REVISE_RQTS} + \text{COST_FIX_RQTS} \right]$$

$$+ \left[\text{COST_PRODUCE} + \text{COST_REVISE_DESIGN} + \text{COST_FIX_DESIGN} \right]$$

WILL

$$\text{COST_REVISE_RQTS} = \sum_i \text{COST_REVISE_RQTS_MOD_I}$$

$$\text{COST_FIX_RQTS} = \sum_j \text{COST_FIX_RQTS_ERROR_J}$$

SE581-049

Figure 5-2 Fault Tolerant Version

of the requirements generation itself, but also the number of errors remaining in those requirements. The total development cost then includes not only the costs of planning, defining requirements, defining design, and producing the product, but must also include the cost of fixing errors, plus revision of the requirements/design/product in response to every requirements change.

The considerations presented to this point have been fairly generic in nature, in that they could describe almost any system or software requirements methodology. However, any further decompositions will necessarily become methodology and tool specific. Figure 5-3 presents an overview of how that might occur. Each step of the methodology is defined in terms of specified inputs and outputs, and then decomposed into activities which can be allocated between manual procedures for using tools, and the capabilities required of the tools, to support the methodology. The interface between the procedures and the tools constitutes the requirements on the man/machine interface. That is, they become the requirements on the syntax for inputs to the tools and for presenting the results back to the user.

The performance indices of the requirements methodology can then be decomposed into costs and schedules for each of the steps, which in turn are decomposable into costs for the man and the costs of the support tools. Thus, each methodology can be described in terms of the sequence of steps of using the tools to achieve a sequence of results, yielding different estimates of costs and rates of errors contained in the requirements. The costs of requirements can now be assessed in terms of this model by tracing the costs associated with each of the outputs of the requirements phase (see Figure 5-2).

The cost of requirements starts with the consideration of the cost of generating the initial requirements (including man and tool costs), but that is only part of the total cost of a methodology. We must be cognizant of the following considerations, as well:

PERFORMANCE RELATIONSHIPS

- COST_RQTS
 $= \text{COST_STEP_1}$
 $+ \text{COST_STEP_2} + \dots$
- $\text{COST_STEP_1} = \text{COST_STEP_1_MAN}$
 $+ \text{COST_STEP_1_TOOL}$
- DIFFERENT METHODOLOGIES HAVE DIFFERENT
 - COSTS
 - * DEFINITION
 - * TRANSLATION
 - * DOCUMENTATION
 - ERROR RATES
 - * NUMBER
 - * COST TO FIX

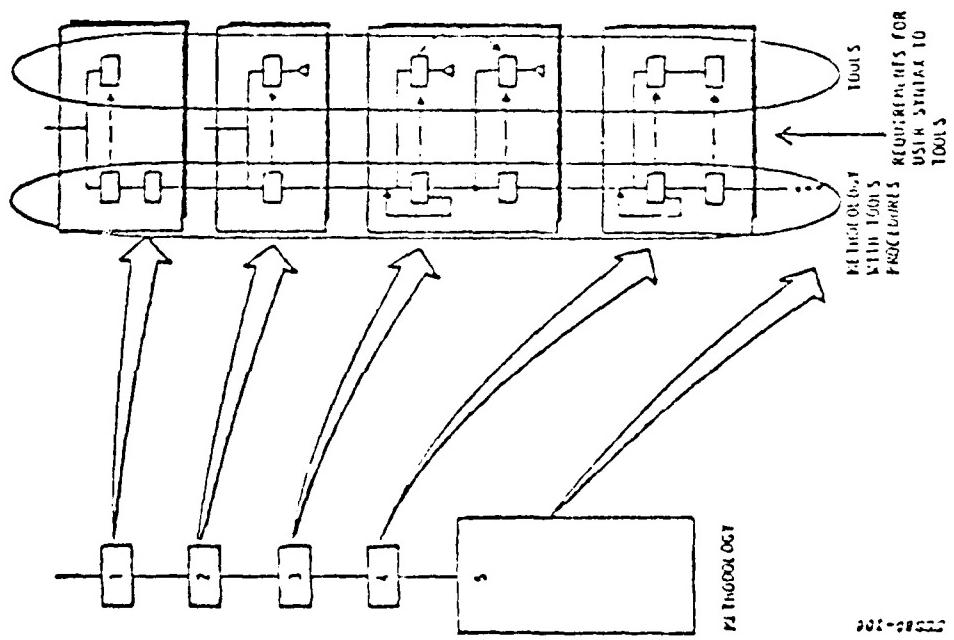


Figure 5-3 Further decompositions are Methodology/Tool Specific

- Because the costs of errors can be very significant (as we shall see shortly), it is generally cheaper to verify requirements so as to discover errors early, rather than to find them and fix them later. Since a methodology may or may not include verification, the requirements generation costs should be separated into two pieces: the cost of requirements definition, and the cost of requirements verification.
- The output of the requirements activity is the requirements used to perform design. If the requirements contain errors, additional requirements and design work will be necessary to identify the errors, correct the requirements, and then correct the design. In turn, when the design is complete and code is being produced and tested, additional code and test costs may occur in response to correcting requirements errors at that stage. To capture these effects, we will assign all design and development costs to fix errors to the requirements activity; if there were no requirements errors, there would be no such costs.
- As modifications to the original problem statement occur, the requirements must be revised to reflect these changes. The costs of modifying the requirements can be attributed to the requirements phase, but the design and production costs are usually allocated elsewhere. Note that as requirements are modified (even if only 1 percent of the requirements are changed), all of the resulting requirements must be re-verified to assure that the requirements changes are consistent with the remaining requirements. Note also that the requirements may be modified hundreds of times, and thus this re-verification cost can become significant if not supported with automated tools.
- Since the requirements are used as the starting point of design, additional work may be necessary to translate them to a form usable in the design effort. Assume for the purpose of this discussion that, if such translation is necessary, the requirements activity should assume half of the costs.
- Since the requirements must be published in an acceptable form, the translation costs into an acceptable documentation format must be included as part of the requirements costs.
- Since the requirements are the starting point of test planning, the costs of translating them so that test case procedures can be designed should also be charged to the requirements activity.

Figure 5-4 presents a summary of these costs and reflects that each of the above costs must be paid for each version of the software system. This includes the initial development as well as all the versions that must be developed during the system's life cycle.

$$\text{TOTAL_COST_RQTS} = \sum_{\text{VERSION_K}} \sum_{\text{MOD_1}} \left[\begin{array}{l} \text{COST_DEFINE_RQTS_1} \\ + \text{COST_VERIFY_RQTS_1} \\ \\ + \sum_j \text{COST_FIX_ERROR_j} \\ \\ + \text{COST_TRANSLATE_TO_DESIGN}/2 \\ + \text{COST_TRANSLATE_TO_TESTABLE} \\ + \text{COST_DOCUMENTATION} \end{array} \right]$$

AMX81-073

Figure 5-4 Cost of Requirements

5.3 THE COST OF FIXING ERRORS VERSUS VERIFICATION COSTS

The Equation of Figure 5-4 can be analyzed from a number of viewpoints. First, consider that the cost of fixing errors grows as a function of when the error is discovered during the system's life cycle. Figure 2-1, shown earlier, presents an estimate of the relative costs of fixing errors as a function of error discovery time derived from data supplied by IBM, GTE, and TRW projects. Suppose we postulate that if we performed only 99 percent of the requirements job, this would result in errors which would be discovered throughout the remainder of the design and implementation phases (ignoring maintenance for the moment). This would require that 1 percent of the requirements job be completed later, as well as the effort necessary to correct the design and code and to retest the system. If we assume that 50 percent of the errors would be found at Preliminary Design Review (PDR), that 50 percent of the remaining errors were found at Critical Design Review (CDR), that 50 percent of the remaining errors were found during development test, and that the remainder were found at acceptance test, then the graph of Figure 5-5 results. Combining the results of this graph with the costs by phase of Figure 2-1, we find that if 90 percent of the requirements effort were successfully completed during the requirements phase, the remaining 10 percent would still result in a doubling of the total requirements cost. If only 80 percent of the requirements were completed, then the cost of the requirements would increase by a factor of five. If we assume the requirements to cost 20 percent of the project budget, then the whole software development project would approximately double. Finally, if only 50 percent of the requirements were completed during the requirements phase, the total project costs would approximately quadruple (i.e., 300 percent overrun).

Another data point on cost can be extracted from TRW's Systems Technology Project experience. On that project, requirements were defined using Engagement Logic, which is a non-automated version of the RSL functional requirements. It required flow charts which identified sequences of processing steps for each input message, showed decision points and algorithms, and described inputs and outputs for each algorithm. It took approximately 30 man-months to produce each new version of the Engagement Logic, and to check it for consistency. This consistency checking was performed manually and took approximately 10 to 15 man-months per version.

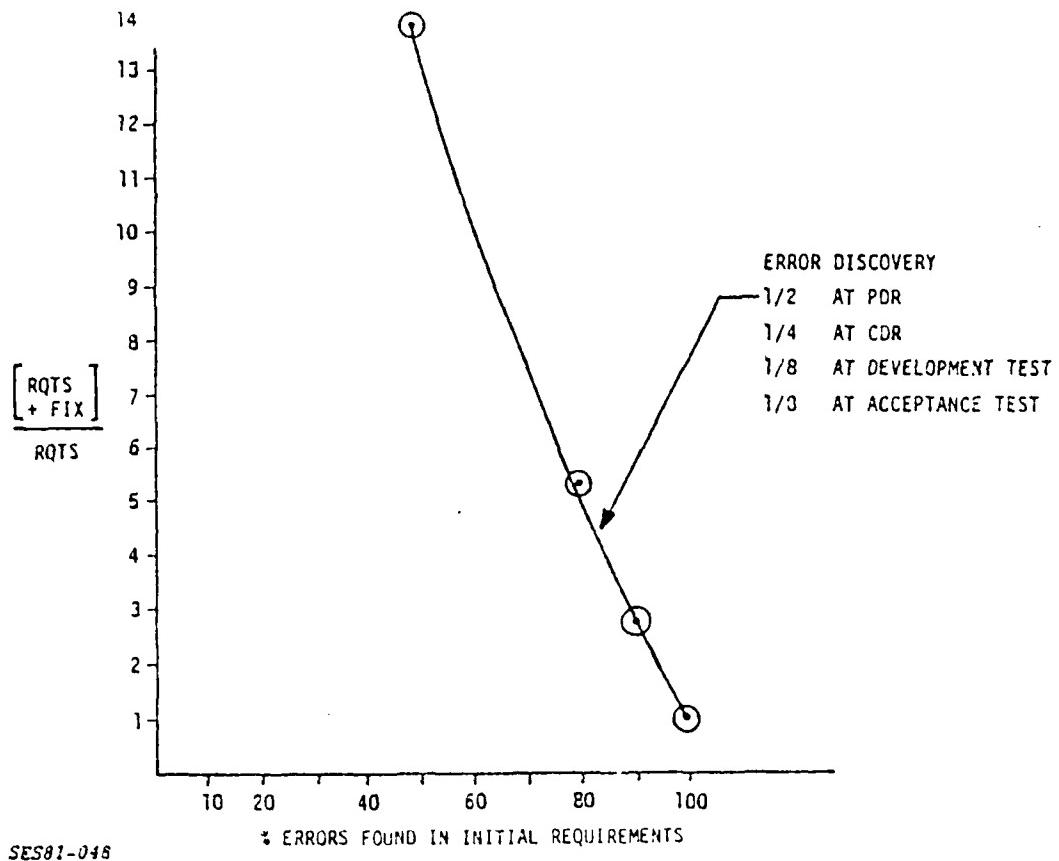


Figure 5-5 Example Analysis

It is interesting to note that this level of validation led to a significantly smaller number of errors at integration and acceptance test time, because the data flow had been thoroughly verified during the requirements phase. It is also interesting to note that the same type of data flow analysis can be performed by REVS on a CDC 6400 in about 10 minutes of processing time at a cost of approximately \$100 for computer time. This shows the significant reduction in the cost of attaining consistency with verification using automated tools.

5.4 SOFTWARE REQUIREMENTS METHODOLOGY EVALUATION

The equation of Figure 5-4 can be used to guide the evaluation of different requirements methodologies in terms of comparing their respective life cycle costs. Table 5.2 provides an overview of the relative costs of four techniques to define requirements for a large project which we will assume will experience at least 10 significant requirements modifications.

Table 5.2 Relative Costs on Large Projects

COST ELEMENT	RELATIVE METHODOLOGY COSTS						
	SREM	(PSL, PSA)	HOS	IORL	SADT	JACKSON	MANUAL
REQUIREMENTS GENERATION	H	M	M	M	M	L	L
REQUIREMENTS VERIFICATION	VL	M	M	M	M	M	H
NUMBER ERRORS	VL	M	M	M	M	M	H
REQUIREMENTS MODIFICATION	M	M	M	M	M	M	H
TRANSLATION FOR DESIGN	?	?	?	?	?	M	?
DOCUMENTATION	VL-M	VL-M	VL-M	VL-M	VL-M	L	VL-M
TRANSLATION FOR TESTABILITY	VL	H	L	M	H	M	H
TOTAL COST	L	M	M	M	M	M	H

The methodologies selected were those of SREM, CADSAT (or other versions of PSL/PSA produced by Dr. Teichroew at the University of Michigan), IORL (produced by Teledyne Brown), HOS, SADT, the Jackson Design Method, and a standard state-of-the-art manual technique to produce a MIL STD 490. The analysis substantiating the estimates of cost are discussed below.

- Requirements Generation: Assume that the cost of defining requirements in an automated form is significant. Then, since SREM requires definition of more information than CADSAT, IORL, HOS, or SADT, one could argue that it would be more costly to use SREM to define requirements; and these would be more costly than the purely manual Jackson or free-form technique. This assumption may be incorrect in that it may underestimate the advantages of a structured methodology for focusing effort on a specific sequence of steps, compared to the degree of effort dissipated without such focus.

- Requirements Verification: The REVS automated analyzers provide a much more thorough analysis for a low cost than can be achieved by hand. CADSAT has a limited number of analyzers (e.g., can check that all data has a legal source and sink, but cannot check for correct data flow because sequences of processing are not representable in PSL). IORL, HOS, and SADT similarly are weaker in automated checks. Thus, if the same level of consistency is to be achieved with these tools as with REVS, additional man-hours must be spent. The amount of effort necessary to manually verify requirements for a manual free-form or Jackson technique will be even higher than for CADSAT and IORL, because of a total lack of automated tools.
- Number of Errors: Error counts will be lowest with SREM because of the automated analyzers; if we assume that the HOS, CADSAT and IORL analyzers will remove specific classes of errors, then these will be lower than the errors in the manual specification. Except with SREM, the data flow type of errors are typically not found until integration test time, where they are very expensive to fix.
- Requirements Modifications: Assume for the moment that the cost to modify requirements using any automated technique is about the same. The cost of modifying a manual specification will be higher because of the necessity to trace all impacts of the change; the automated tools of the other techniques should assist the modification process. The cost of SREM is medium, instead of high, because of the additional traceability techniques used during the requirements generation activity.
- Translation to Design: This is left as an open issue because of the lack of a standard design technique, except for the Jackson Technique.
- Documentation Costs: These costs can vary from very low to medium using automated techniques (depending on whether the automated documentation produced is in acceptable format), and varies from low to medium for manual techniques, depending on desired format and assuming that the analysis has been performed. For example, an existing RSL data base for a medium sized software project has been translated into a standard 8-5 specification in a few weeks with about 3 to 4 people. This is small in comparison to the time to generate such requirements manually.
- Translation to Attain Testable Requirements: Since RSL, by design, produces testable requirements, the effort to translate these into a form for performing test planning is very low. IORL has, as one of its

characteristics, the statement of stimulus/response requirements, but does not completely identify the information to be measured, and the accuracy requirements to be met in testable terms, so that additional work is necessary. The great weakness of both CADSAT, SADT and a standard manual requirements technique is that the resulting requirements are not testable, thereby requiring a considerable effort to translate them into a testable form. The HOS technique results in testable functional requirements, but does not address explicitly issues of accuracy.

- Total cost: If we assume for a medium to large project that several requirements modifications will take place, then the costs of requirements generation and documentation will be dominated by the costs of performing verification and the costs of fixing all requirements errors. This leads to the conclusion that, since SREM identifies and reduces requirements errors more effectively than other techniques, the overall costs will be lower than those of CADSAT, HOS, SADT and IORL, and that these will have lower overall costs than a manual technique.

The conclusions of this analysis are a function of project type, project size, and the design quality of the software. If the cost of an error in the software is not large (i.e., can be lived with, or fixed when discovered), then the needed level of verification may be lower. If the size and complexity of the software is small, and there will be no significant modifications to the requirements, then an individual analyst may be able to perform verification in his head as effectively, and with little more effort than with automated tools. However, for large software applications, or those requiring significant design quality, the use of SREM becomes advantageous.

5.5 CONCLUSIONS

At least two interesting implications can be derived from this approach for defining overall the cost of requirements generation:

- Methodology cost cannot be properly evaluated on the basis of a one time application during the requirements phase. As we have seen, a significant portion of the cost of a requirements methodology is a function of when the errors are discovered, and which techniques were used to correct them. A project which has had detailed reviews at PDR, CDR, etc., will probably find more requirements errors earlier than one which does not. In addition, the cost of translation of requirements into a form useful for software design, test planning, and project documentation is project dependent. Finally, the required quality of software and degree of risk acceptable to the project of late delivery due to correction of errors detected during acceptance testing is also project dependent. Thus, without consideration of these parameters, no true evaluation of the cost and relative merit of a software requirements methodology is possible.
- The leverage of automated tools on the cost of requirements generation comes from a simple fact: even though only 1 percent of the requirements may be modified, all of the remaining requirements must be verified, and the documentation must be modified correctly to reflect the changes. Thus, since modification, verification, and documentation are repeated many times on real projects, tools which efficiently verify and document the changes have a higher payoff. Finally, because the total cost of the requirements is lower with automated tools, it may be possible to use some of the extra funding to search a wider design space in order to reduce the operational costs of the software, or to increase the overall effectiveness of the software itself.

6.0 ASSESSMENT AND RECOMMENDATIONS

In this section we will first consider the applicability of SREM to Army management information systems. Next, we will discuss the kinds of problems we encountered that may be typical to SREM verification efforts on management information systems. Finally, in light of the problems found, we will recommend enhancements that would increase SREM's capability for requirements formulation and requirements verification.

6.1 APPLICABILITY OF SREM TO ARMY MANAGEMENT INFORMATION SYSTEMS

The essential functions of a management information computer system are to support the collection, correlation, analysis, retention, and display of information and to support analysis, decisions, dissemination, and other activities necessary to perform the military mission. While the range of systems and the support missions may be quite broad, most management information systems have similar generic characteristics that distinguish them from weapon systems with embedded computers -- the type of system for which SREM was initially conceived.

One of the key characteristics of a management information system is that it cannot be totally automated. There is a man in the loop, and he is there to accomplish appropriate data entry, make situation-dependent decisions, apply judgement, and permit appropriate responses to a variety of inputs that cannot be fully anticipated. Computer support is provided where fixed procedural operations can be defined and where automation is needed to eliminate drudgery, handle load requirements, and/or net response times. Since a fixed algorithm for performance of the mission cannot be specified except, perhaps, in broad generalities, the user is given specific automated capabilities and the ability to apply them in sequences and combinations of his choosing to perform his particular job. Heavy emphasis on human engineering and man/machine interface is necessary to make the system effective and responsive.

The variety of different message types entering and leaving the system is large, and formats may be diverse. Data base structures are typically large, varied, and complex. Cross-correlation of data structures is extensive, collections of data elements may be used in multiple roles, and data base management may be complicated by security requirements.

While requirements for application software embedded in weapon systems can be expressed in terms of the engagement logic and operating rules of the weapon system, it is often difficult to specify requirements for management information system software in similar terms. First, the data processor capabilities for management information systems are often multi-purpose and their actual use is determined by the user within a particular situation context. Second, the possible variety of stimuli to such systems is often so large that complete definition of all possible uses may not be possible.

Even in consideration of these differences, SREM can be, and has been, applied to systems with the characteristics of management information systems. Examples of past applications of SREM by TRW to such systems are outlined in Table 6.1.

Table 6.1 Description of Management Information Systems to Which SREM Has Been Applied

SYSTEM	TYPE APPLICATION	DESCRIPTION OF SYSTEM OR EFFORT
CV-ASWM (U.S. NAVY)	VERIFICATION	THIS SYSTEM SUPPORTED THE COMMAND AND CONTROL OF ALL ASPECTS OF ASW FROM A CARRIER. IT INCLUDED INFORMATION ON ALL THREATS; ALL AIR, SURFACE, AND SUB-SURFACE FRIENDLY LOCATIONS; ALL ASW-SYSTEM LOCATIONS (SUCH AS SONOBUOYS); AND INFORMATION ABOUT ALL RESOURCES THAT COULD BE BROUGHT TO BEAR ON THE ASW EFFORT
NAVDAC BUSINESS DATA PROCESSING (U.S. NAVY)	DEMONSTRATION	NAVDAC IS THE NAVAL DATA AUTOMATION COMMAND, WHICH HAS RESPONSIBILITY FOR ALL "BUSINESS" DATA PROCESSING (NON-TACTICAL SOFTWARE). SREM WAS APPLIED TO TWO SAMPLE PROBLEMS TO ASSESS THE APPLICABILITY OF SREM. THESE WERE THE SURFACE WARFARE DATA INTERPRETATION SYSTEM (SWARDIS), AND THE CONTINGENCY AMMUNITION REQUIREMENTS AND SUPPORTABILITY SYSTEM (CARESS).
IHAWK/TSQ-73 (U.S. ARMY)	DEMONSTRATION	THE IMPROVED HAWK SYSTEM COMMUNICATES WITH THE AN/TSQ-73 MISSILE-MINDER COMMAND AND CONTROL SYSTEM. THIS STUDY PRODUCED AN RSL DATA BASE FOR THE IHAWK SOFTWARE REQUIREMENTS INCLUDING THAT RESULTING FROM INTERFACE WITH AN/TSQ-73.

AN/TSQ-73

6.2 SREM APPLICATION TO TYPICAL MANAGEMENT INFORMATION SYSTEMS

Application of SREM to these systems, as well as to the MOM DFSR, presented certain unique application challenges that caused some additional methodology concepts to be addressed. These will be discussed in following paragraphs.

6.2.1 CV-ASWM Application

One of the most unique application challenges had to do with how to define the concept of a PPI scope filled with symbology, all of which could move, could be individually blinked, could be at several levels of display intensity, could be individually deleted, added, or temporarily inhibited, and which required various classes of items (such as all surface vessels) to be inhibited. Until that point in time, the concept of SREM dealt with the arrival of a single MESSAGE, determination of appropriate processing, and the output of resulting MESSAGEs to other subsystems of the weapon system, but not to display consoles. Now, for the first time, we had an operator in the loop and a complex visual display to contend with.

Under old considerations, each change to an object on the screen would have been a new output MESSAGE to the CONSOLE. This would have been exceedingly cumbersome and the resulting R_NETs would have been difficult to understand. Our solution was to create a display ENTITY_CLASS that contained all the information about each item on the scope, plus all the data required for their display control. Thus, for each instance in the ENTITY_CLASS there were control DATA items to:

- Indicate if the symbol was blinking.
- Describe the symbol's level of display intensity.
- Indicate whether or not the symbol was inhibited.

In this way, deletion of a displayed item could be accomplished by having its instance of the display ENTITY_CLASS DESTROYED BY an appropriate ALPHA. Similarly, new objects were added to the display ENTITY_CLASS, when appropriate, by being CREATED BY a different ALPHA. Any change in the control data for the display was also indicated in this ENTITY_CLASS by changing the value resident in the appropriate Control DATA item.

Thus, at any instant of time, the appropriate conditions of each item in the display ENTITY_CLASS were defined by the value held by the control

DATA in each instance of the display ENTITY_CLASS. We then defined the display processing as a self-enabling R_NET which was initiated when the system was turned on, and which re-enabled itself using an EVENT at the display refresh rate. What the R_NET did each time it was enabled was to access each instance of the display ENTITY_CLASS which was not inhibited (via a FOR EACH node) and used the DATA values of that instance in an output MESSAGE to the console.

A new challenge was to describe the controls for accessing the system's data base, and for allowing changes to be made and examined at the console without actually changing the data base. This was done by establishing a "copy" of the data base for use (which meant temporary establishment of a second ENTITY_CLASS) in the RSL data base to represent the copied one for operator use.

It was also necessary to maintain information identifying the operating mode of each console, since certain functions were available only while the console was in a particular mode. This was accomplished by establishing an ENTITY_CLASS for console control with the requisite control DATA. This also required that the first node on each R_NET which was ENABLED BY the INPUT_INTERFACE from the consoles be checked to determine the console's operating mode to determine whether the MESSAGE that had just entered was legal for that mode.

These, and other new challenges, had to be met in applying SREM to the CV-ASWM. All of them were overcome with a solution within the capabilities and constraints of the existing RSL and REVS capabilities.

6.2.2 NAVDAC Application

It was concluded in the NAVDAC study that:

"While some modification to the REVS tools and SREM methodology are indicated, the approach developed originally for tactical problems appears to be adaptable to BDP (Business Data Processing) needs."

NAVDAC Business Data Processing can be divided into four major functional categories:

- Logistics - procurement, maintenance and transportation of military material, etc.
- Financial Management - accounting, appropriation, financial projections, etc.

- Administration - management information, executive functions and decisions, etc.
- Personnel - historical and current personnel data.

Because of the close similarity of the NAVDAC requirement to a management information system, such as the MOM DFSR, we have excerpted the portion of the NAVDAC Final Report that describes and evaluates the use of SREM and the limitations experienced. Systems to which SREM was applied are first described:

"3.1.1 Surface Warfare Data Interpretation System (SWARDIS)"

The SWARDIS is the initial increment of an ADP system that ultimately will provide the staff of the Deputy Chief of Naval Operations (Surface Warfare) (Op-03) with all ADP capabilities required to support the Op-03 functions and procedures involved in the preparation of the Navy's Program Objectives Memorandum (POM). The objective of the SWARDIS is to convert data from the Department of the Navy Five Year Program (DNFYFP) files maintained by the Navy Cost Information System (NCIS) to a file structure that conforms to the Op-03 resource management concepts so that the Op-03 staff can prepare reports using NCIS Five Year Defense Program (FYDP) data couched in Op-03's management terminology. System design concepts allow the user staff to specify whatever data file structure is deemed appropriate, to define the user staff organization and responsibilities (cognizance) for user programs, and to establish up to eight means of "prioritizing" the resource requirements for which the user is responsible."

"3.1.3 Contingency Ammunition Requirements and Supportability System (CARESS) Model"

The CARESS model, developed for the Weapon Logistics Readiness Division (J42) of CINCLANTFLT, calculates the conventional ordnance needed to support a contingency situation or a specific Operational Plan (CPLAN) and determines whether or not the Navy can meet the needs on a daily basis. The model determines and allocates task force ammunition requirements on a day-by-day basis, and then tests, for each day, the availability of the required ordnance and the ability of ammunition ships assigned to the task force to deliver the ammunition when it is required. If a plan is not supportable, the model may indicate some of the factors responsible. If the desired ordnance is not available, attempts are made to substitute ordnance from compatible weapon stocks."

"..... the CARESS REVS data base is comparatively large. Despite its size and complexity, we developed a comprehensive processing structure for the problem within four weeks after start, and had identified major critical issues, without previous exposure to the application area."

"3.2 ASSESSMENT OF APPLICABILITY

This section presents TRW's assessment of SREM applicability to Navy BDP problems, based on evaluation of representative documentation and development of the selected sample problems."

"3.2.1 Project Size and Complexity

Table 6.2 summarizes the Navy BDP problems reviewed, presents an estimate of complexity, and indicates the applicability of SREM supported by the automated tools. The complexity of the system was estimated by TRW using factors defined by DoD Standard 7935.1-S.

Table 6.2 Survey of SREM Applicability to NAVDAC Applications

SYSTEM ACRONYM	DOCUMENTATION EXAMINED	SREM EXAMPLE DOCUMENT	SREM APPLICATION	COMPLEXITY ESTIMATE*	REMARKS
SEACUP II	NARDAC, WASH. D.C. 63F1002, JM-01 (MAR 79)	NO	YES	30-33	<ul style="list-style-type: none"> • COMPLICATED LOGIC • REQUIRES OPERATIONAL PLANNING AND MODELING EXPERTISE • MORE COMPLEX THAN CARESS
SWARDIS	NARDAC, WASH. D.C. 63F1002, FD-01 (JUL 78)	YES (APPENDIX A)	YES	28-31	<ul style="list-style-type: none"> • REQUIRES USER LANGUAGE DESIGN EXPERTISE AND ANTICIPATION OF DATA ENTRY ERRORS
CARESS	NARDAC, WASH. D.C. 71L1001, UM-01A (NOV 78) NAVCOSACT 53E21ZC, FD-01 (SEP 75)	YES (APPENDIX B)	YES	26-28	<ul style="list-style-type: none"> • COMPLICATED LOGIC • REQUIRES CONTINGENCY MODELING EXPERTISE
LAPIS	NAVCOSACT 02A004, FD-01 (JUN 73)	NO	PARTIAL	24-26	<ul style="list-style-type: none"> • UNDEFINED USER QUERY REQUIREMENTS MIGHT BE AREA WHERE SREM IS NEEDED
DEMS	NAVCOSACT 02A066, FD-01 (MAR 76)	MANUAL	NO	23	<ul style="list-style-type: none"> • ROUTINE DOMESTIC DATA CONSOLIDATION AND REPORTING SYSTEM
MRAAC	NARDAC, WASH. D.C. 61M1012, FD-01 (JAN 79)	NO	NO	20	<ul style="list-style-type: none"> • SIMPLIFIED USER INTERFACE WITH LIMITED MENU, PROMPTING, TEMPLATED INPUT
JALPES	NARDAC, WASH. D.C. 02A074, FD-01 (SEP 77)	25% EXTENSION EXPERIMENT	SYSTEM LEVEL	16	<ul style="list-style-type: none"> • EXTENSIONS WOULD BE USEFUL FOR SYSTEMS WITH MORE COMPLEX TRADEOFF ISSUES

*TRW ESTIMATE USING DOD STANDARD 7935.1-S CRITERIA

The break-point for full SREM application would appear to be a complexity level of about 25. More complex projects would benefit from full use of the automated tools. Less complex projects would benefit from manual utilization of the (SREM) concepts down to a level of 20 to 21. Below this level, benefits would decrease to a negligible value for software requirements, although extension applications at the system level might be worthwhile.

Two factors should qualify these estimates. First, we estimated complexity values without previous experience in using this standard. Experienced NARDAC personnel might derive higher or lower values for these projects. The break-even point should be calibrated according to their assessment of complexity.

Second, the DoD Standard complexity factors do not include a direct measure of the logical complexity of the application. Nor do they differentiate between the complexity of the user's information processing problems and the problems of physical implementation. The

logical complexity may be inferred from the number of people assigned to the project and its costs, but other contributing factors also influence those variables, including the overall size of the application.

Presuming logical complexity roughly equivalent to SWARDIS or CARESS, applications roughly one-tenth of their size could benefit from use of SREM. Indeed, one of the significant advantages of SREM is that it reveals the true size and complexity of problems, often underestimated by purely verbal descriptions."

Certain RSL extensions, which are easily made because of RSL's extensibility feature were found to be required. Of more interest, however, are the limitations that deserve REVS enhancements. These include the capability:

- For an additional value for the attribute TYPE as it applies to DATA to provide for ALPHANUMERIC DATA. The current DATA TYPES include INTEGER, REAL, BOOLEAN, and ENUMERATION. REVS would need to be changed to represent this TYPE of DATA in simulations, and its use to qualify an OR node branching decision would have to be worked out.
- For complex ordering of an ENTITY_CLASS, ENTITY_TYPE or FILE on a multiple set of keys (primary, secondary, tertiary, etc.). Currently, one can define the transition from one ordering to another only by describing a long, detailed sequence of processing steps.
- For a sequential FOR EACH mechanism to perform a processing step on each instance of a set, in a specified set order.
- To express WHILE and UNTIL conditions for a FOR EACH. The current FOR EACH is intended to specify that all instances be processed, and if this is to be done only until some state is reached, it currently cannot be so defined, structurally.
- For a SELECT capability on a MAXIMUM or MINIMUM value of the DATA in some instance of an ENTITY_CLASS, ENTITY_TYPE, or FILE. This capability does not now exist, but the need for it frequently occurs in management information systems.

The above REVS changes, of course, would require METHODOLOGY modifications, as well.

6.2.3 IHAWK/TSQ73

This was a large demonstration effort accomplished in 1977-78 to determine whether SREM was as applicable to tactical missiles as it was to BMD systems. The conclusion on applicability of RSL/REVS for the study are shown in Table 6.3 and, as indicated, certain RSL enhancements were recognized as beneficial. These are briefly discussed below.

Table 6.3 SREM Applicability to the IHAWK/TSQ73 Application

- THE MAJOR DIFFERENCES BETWEEN IHAWK AND PREVIOUS BMD SYSTEM CONSTRUCTS, FROM THE STANDPOINT OF DIFFERENT TYPES OF SOFTWARE REQUIREMENTS, ARE:
 - OPERATOR COMMAND ORDERS
 - DISPLAY PROCESSING
 - C³ DATA LINK PROCESSING (MESSAGE SEQUENCING, DATA LINK CAPACITY, INTERIM OUTPUTS)
 - SUBSYSTEM SENSING (CONTINUOUS INPUT INTERFACES)
- RSL WAS ABLE TO SPECIFY ALL CONDITIONS IMPOSED BY IHAWK (WITHOUT USE OF THE EXTENSION SEGMENT)
- SOME CHANGES TO THE LANGUAGE WERE IDENTIFIED THAT WOULD PROMOTE EASE OF EXPRESSION AND USER CONVENIENCE
 - NON-TERMINAL OUTPUT INTERFACES
 - CHANGES IN ENUMERATED DATA
 - STANDARD FUNCTIONS IN CONDITIONALS
 - ADDITIONAL RSL WRITE/MODIFY AIDS

AMAS1-156

6.2.3.1 Non-terminal Output Interfaces

The need to specify the output of a sequence of MESSAGES occurred for the transmission of MESSAGES over a data link. An indirect approach had to be taken to express this type of requirement in RSL. The information that MAKES the MESSAGE had to be saved in FILES and an R_NET that accesses the FILES and determines the MESSAGE to transmit had to be developed. By allowing OUTPUT_INTERFACE to serve as non-terminal nodes in a structure, such as shown in Figure 6-1, this type of requirement would be easier to specify and more understandable in RSL. Currently, an OUTPUT_INTERFACE terminates a branch of processing.

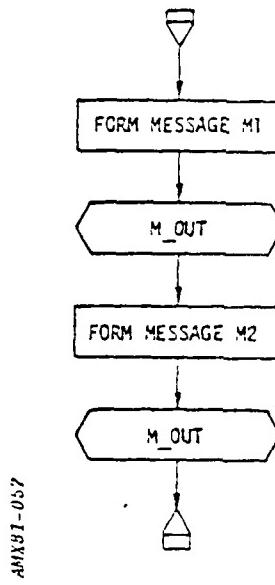


Figure 6-1 Illustration of Sequential MESSAGE Output Using Non-Terminating OUTPUT_INTERFACEs

6.2.3.2 Changes in Enumerated DATA

The use of enumerated DATA provides a clear expression of the meaning of DATA values. For example, it is clearer to state that the values for SENSORS are "IPAR, ICWAR, IHPI" than to state the values as 0, 1, or 2. There are two changes pertaining to the use of enumerated DATA that would make it more useable:

- Allow the use of the value of an enumerated DATA item in the standard conditional. For example:

```
SELECT ENTITY_CLASS TRACK SUCH THAT (TRACK_TYPE =
REMOTE)
```

WHERE

DATA TRACK_TYPE is of TYPE ENUMERATION with a RANGE "LOCAL, REMOTE".

Current rules only allow either numerical values in the conditional statement such as:

```
SELECT ENTITY_CLASS TRACK
```

```
SUCH THAT (TRACK_NR = 84)
```

or the comparison of two named DATA items in the conditional statement, such as

```
SELECT ENTITY_CLASS TRACK  
SUCH THAT (TRACK_NR = TRACK_NR_IN).
```

Thus, an enumerated value (which is defined as a value stated in words) cannot currently be used in a conditional statement.

6.2.3.3 Allow Standard Functions in Conditional Statements

Currently, the reference of standard math functions in conditional expressions is not allowed in RSL. For systems such as IHAWK (that perform a lot of geometric computation) a capability to use standard functions in conditional statements would provide a more natural expression of software requirements. Examples are:

- IF (ABS(RANGE) > RNG_LIMIT).
- SELECT TRACK SUCH THAT (SIN(AZ) > SECTOR).

6.2.3.4 Provide the Capability to Assign a Particular Attribute and/or Relationship to a Collection of Elements

An example of how this might be useful is when a set of modifications are to be entered into the requirements data base as the result of a change to the software specification on which the requirements data base is based. RSL has an attribute called VERSION which allows changes to be recognized by adding this attribute to all new data base entries that are the result of such a change. Currently, each such element has to be stated as the subject element to which this attribute VERSION can then be assigned. What is being sought here is a method of providing the replication of the VERSION attribute in a more automatic manner to all effected elements.

6.2.3.5 MCM DFSR Application

The difficulties we have experienced with the MCM DFSR have primarily been those of scale, as were described in Section 3.0. We also experienced several of the limitations outlined above, plus two which were apparently not encountered on other efforts.

The first enhancement that would have been useful would be the capability to show the SELECTION of an instance of an ENTITY_CLASS or ENTITY_

TYPE which possesses the DATA item that has the next higher (or lower) value to that DATA item in the currently SELECTed instance of the ENTITY_CLASS or ENTITY_TYPE. To describe this requirement under current methodology is cumbersome.

A second area we encountered, which is unexpressible within the current structure capabilities, was the Real-Time requirement XMH starting with DLT 619. The essence of this part of the requirement is that when CARD_DSG_CD_SAMS is input with a value of A, and INQ_ACT_CD has a numeric value, a look-up table is accessed where a particular data element name is linked to the INQ_ACT_CD. The intent is to use the data name to access its equivalent data name in a previously specified file; either the WORF or the TPR. This would require a memory map showing the location of the desired data element so that the data element at that location could be accessed and its value displayed.

Conceptually, access via data location is at a different level than that used to describe normal application requirements. For example, in RSL, an instance of an ENTITY_CLASS may be accessed such that some DATA item in the ENTITY_CLASS possesses a particular boolean condition. This may relate to a comparison of the value in the DATA item to a real integer number, or the boolean expression may be true for some comparison of values of two DATA items. Both of these conditions were illustrated in Paragraph 6.2.3.2 for the ENTITY_CLASS: TRACK. The meaning for a statement of the type such as (TRACK_NR = TRACK_NR_IN) is that the value resident in TRACK_NR is equal to the value resident in TRACK_NR_IN. The portion of the XMH requirement described above requires that the equality be accomplished by comparing the DATA name in the look-up table to the equivalent name in the ENTITY_CLASS of interest. Because this type of comparison is not possible within the current SREM concepts, it is not possible to describe that portion of the XMH process on a structure. Fortunately, the designers of SREM contemplated that not every software requirement which might arise could be defined on a structure. Thus, the element: UNSTRUCTURED_REQUIREMENT was included as an RSL element for such an occurrence. An UNSTRUCTURED_REQUIREMENT was used to define this MCM CFSR requirement in the requirements data base. Table 6.4 shows the limitations experienced in various SREM application as described above.

AD-A100 720

TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA
APPLICABILITY OF SREM TO THE VERIFICATION OF MANAGEMENT INFORMATION--ETC(U)
APR 81 R P LOSHBOUGH, M W ALFORD, J T LAWSON DAHC26-80-C-0020
UNCLASSIFIED TRW-37554-6950-001-VOL-1 NL

F/6 9/2

3 OF 3
40 A
1000720

END
DATE FILMED
7-81
DTIC

ANAK97-055

RSL/REVS ENHANCEMENTS	SREM APPLICATION			
	CV-ASW	NAVDAC	TIAWK/TSQ-73	MOM DS/R
1. DATA ATTRIBUTE TYPE VALUE: ALPHANUMERIC	•	•	•	•
2. COMPLEX ORDERING OF ENTITY_CLASS, ENTITY_TYPE, AND FILE	•	•	•	•
3. SEQUENTIAL FOR EACH	•	•	•	•
4. FOR EACH WHILE (OR UNTIL)	•	•	•	•
5. SELECT ON MAXIMUM OR MINIMUM VALUE OF DATA ITEM IN AN ENTITY_CLASS OR ENTITY_TYPE	•	•	•	•
6. NON-TERMINAL OUTPUT_INTERFACES	•	•	•	•
7. ENUMERATED DATA USE IN CONDITIONAL EXPRESSIONS	•	•	•	•
8. MATH FUNCTIONS USE IN CONDITIONAL EXPRESSIONS	•	•	•	•
9. EXPRESSION OF THE MEMORY LOCATION OF A DATA ITEM	•	•	•	•
10. SELECT ON THE NEXT HIGHER OR LOWER VALUE OF A DATA ITEM	•	•	•	•

Table 6.4 Enhancements needed for Various SREM Applications to Systems with Characteristics Similar to Management Information Systems

6.3 SREM ENHANCEMENTS

Table 6.4, previously presented, outlines certain of the enhancements we would have found useful in the application of SREM to various kinds of non-BMD systems. We believe that all of the enhancements are worth considering with the exception of Item 9. The concept of application of SREM to the level requiring manipulation of data based on memory locations is at a level of detail not contemplated for SREM. Further, requirements of this kind can be stated textually much more easily and clearly.

The other nine enhancements would be useful to ease application efforts, and to produce clearer understanding for the software designer, than is the case using current concepts. Based on our efforts to date, however, we believe we can work around these limitations satisfactorily, even if enhancements were not pursued. Although it would be necessary to investigate the scope of these enhancements to determine their impact on REVS translation, RADX, and simulation functions, it is fair to say that they probably are not trivial.

One additional capability would have great utility in the software requirements development process; the capability to automatically produce the software requirements document in an appropriate format from the RSL data base. Currently, various RADX commands can be used to present elements of the data base in various ways. The sample regeneration of requirements for a portion of the MOM DFSR in Appendix B is an example of the current capability. Although RADX provides a lot of flexibility it cannot produce a specification in a current Army format. It would be feasible, however, to add this capability. Of all the possible improvements discussed, a "specification generator" of this type is probably the most important and would add a truly useful capability for Army SREM users.

6.4 A COMPLEMENTARY TOOL FOR SREM

As we have discussed, SREM is a methodology which provides the approach, the language, and the automated tools to 1) define the software requirements from the system level requirements, or 2) verify the adequacy of an existing software requirement. If it is the Army's intention to utilize the full power of SREM by applying it to define software requirements (as opposed to only its verification role) then TRW's Performance and Cost Analysis Model (PERCAM) is a logical complementary tool at the system level.

PERCAM's primary role is the investigation of overall operating considerations for computer systems, to identifying bottlenecks caused by resource constraints, and to provide a quick-response tool for evaluating the capabilities and costs of different arrangements or types of processing equipment to accomplish an overall requirement. Investigation might consider processing loads over time, and the adequacy of throughput, queue loading, memory capacity, communication data rates, etc., to handle these loads.

Using results of these simulations, PERCAM would present a comparison of the alternative processing arrangements, computer hardware candidates, and the resulting costs of the approaches under consideration. It could suggest the best arrangement of core versus off-line memory, the number and location of operators needed, and where critical, the speed of response to operating loads. Its relationship to SREM is through the identification ORIGINATING REQUIREMENT that must be imposed on the software for the desired system.

PERCAM also is useful in assessing operational changes under consideration which would impact processing on an existing system. It provides suggestions for alternative computer and/or communication resources to handle the changed processing loads or throughput requirements, or the best reallocation of existing resources to support the changes.

Of course, there are many ways of solving problems of this type. The purpose of the remainder of this discussion is to describe PERCAM and to highlight its rapid turn-around capability, its low cost of application, and the simplicity of its use, when compared to other tools which exist for analysis of this type.

Analysis tools for use in conducting tradeoff studies and in evaluating system elements, complete systems and system families in a dynamic simulation are categorized in two general types. The first type comprises simple, easy-to-use models (usually based on linear programming, game theory, etc.) that accommodate only gross system descriptions and cannot directly measure the effect of variations in system components, organizational structure, or scenario. The second type is characterized by complex and detailed time-dependent, high-fidelity system simulations which are inflexible, require large and detailed data bases, a high degree of user skill, and which provide highly detailed results.

The research and development planning process usually calls for the rapid evaluation of candidate systems and system components under conditions which represent current or projected system capabilities and scenarios. The necessity to quickly evaluate the effects of variations in component performance, cost, and system configuration preclude, in many cases, the use of the simpler models during this stage. While it is possible to use a large-scale conventional simulation to tradeoff studies involving changes in system configuration and in system component characteristics, the required setup time and the computer and manpower resources is frequently much larger than the plan development task could support.

An analysis tool to assess these early requirements should provide quick turn-around analyses, be easily adaptable to a variety of problems and to a variety of requirements fidelities, incorporate the ability to represent and to vary scenarios and operational doctrines, permit the presentation of interaction of the proposed system with its operational environment, and incorporate the model which permits comparative effectiveness evaluation of dissimilar systems. The TRW systems analysis tool, PERCAM, has been developed to meet these requirements.

PERCAM is a simulation methodology which provides a set of software tools to automate simulation generation directly from Event Logic Trees (ELTs) -- a graphic model of system logic and control. The key elements of PERCAM simulation methodology are: the ELTs modeling technique, the PERCAM preprocessor and underlying simulation superstructure. Systems architecture and software modeling is accomplished with the preprocessor which uses a library of FORTRAN components similar to macros to construct simulation models according to user inputs. The structure of the simulation is

modular, flexible, and specifically designed to provide high visibility to system engineers and designers. Changes to system parameters, individual system models, or system configuration are easily accomplished. PERCAM is operable on the CDC and VAX 11/780 computer systems.

Event Logic Trees (ELTs), illustrated in Figure 6-2, are structured graphical representations of the sequence of actions performed by a system in its operating environment. The ELT consists of a series of linked functional blocks which describe the operational paths the system may take to reach any number of termination points. Branching within the tree is controlled by various decision nodes.

The transformation step from the ELT description to the PERCAM input description is straight-forward and, in fact, can be considered as a "cook-book" approach. Since the ELT is constructed with specific components in mind, each point on the ELT corresponds to a particular component and thus, the specific data for that component will be defined by the user when the ELT is transformed into a computer processable input description.

PERCAM provides a quick response, low cost capability for exposing system operational issues via system simulation. The PERCAM methodology facilitates quick assessment of degraded nodes of system operation, makes possible the determination of the key operational characteristics of alternate system configurations, provides a bulk-filter to determine critical issues and high impact design tradeoffs, and at the same time provides insight into peak loading effects and throughput bottlenecks.

Flexibility and long-term growth potential are an inherent feature of the PERCAM concept. Since the component library can easily be updated and modified, component availability can be upgraded to keep pace with evolutional developments. New PERCAM components can be added to represent new and innovative logic concepts or to simply chain together a set of often utilized macros.

Growth potential for PERCAM based simulation is significantly enhanced due to the high level self-documenting communication qualities of ELT descriptions. Systems Analysts are freed from the fears and complexity of large, slowly evolving customized simulation codes whose developers are no longer available.

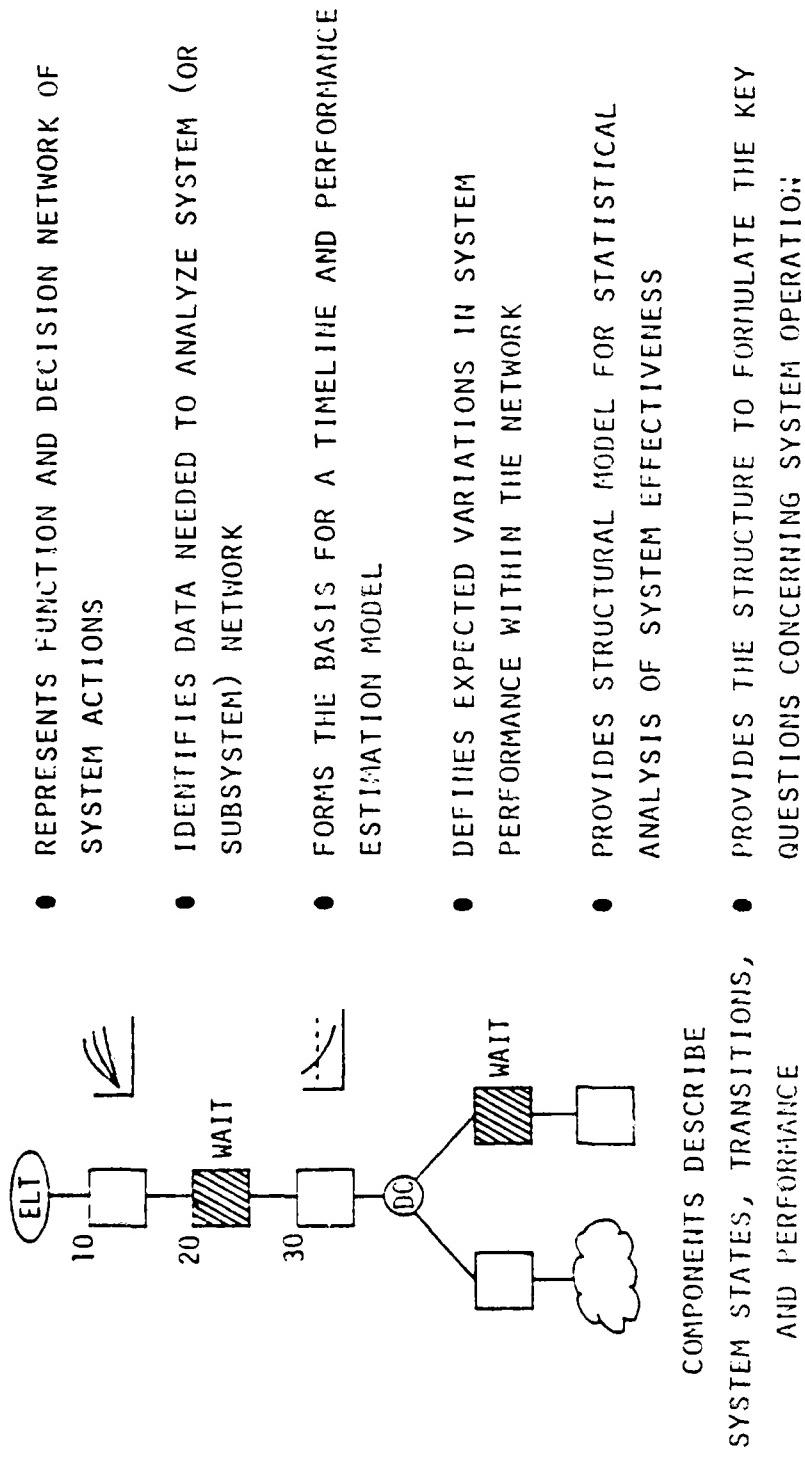


Figure 6-2 Event Logic Trees, The basic for Automated Analysis

6.5 SUMMARY

The major observation derived from the SREM application to the MOM DFSR was that the current definition of RSL and REVS was adequate in scope and flexibility to verify all stated requirements. Some extensions of the language were found appropriate to meet certain management support functions (such as those related to Trouble Reports), and to provide certain RADX documentation to produce information similar to that presented in Annexes A, B, C, and D. This latter extension was primarily for the purpose of illustrating that RSL extension can provide the flexibility to produce information, such as DATA names, in various contexts, each with their own particular relationships and attributes. Even though exact formats cannot currently be produced directly by RADX commands, all the relevant information can be provided, as illustrated in Appendix B.

As has been experienced on previous SREM applications, we found that RSL forces a level of completeness and specificity that is more technically consistent than can be obtained with traditional English specifications, and it places more formality and discipline into the requirements development activity than is possible with a manual approach. This was reflected in this verification effort by the high quantity of deficiencies found in the MOMs DFSR. The added effort initially required with SREM will be more than returned during the software design, implementation, and test phases, and thus reduce the total cost of the software acquisition. There is also a body of opinion that the cost of the requirements development itself may actually be less expensive when SREM is applied for software requirements development. This is because the rapid, consistent printouts reduce the need for coordination and manual consistency checking, and the methodology itself efficiently focuses the necessary activities.

REVS contains a variety of user functions that are available for the specification, analysis, simulation, and documentation of software requirements. All of these functions, except the simulation function (which was not required under this contract) were exercised during the REVS application to the MOM DFSR. Since the majority of the options of each function were necessary to develop and state the MOM DFSR requirements, this effort provided a detailed test. As in previous efforts, it was found that certain enhancements would have assisted in assuring that the SREM application

was not only more efficient, but also more understandable for the software designer's use.

The importance of utilizing an organized approach (a methodology) with the supporting automated capability to check the consistency, completeness, clarity, logicalness, testability, and traceability cannot be over-estimated. The need to attain good software requirements prior to software design and coding is understood by nearly all involved in data processing. However, it is only recently that sufficient interest has been demonstrated to cause a variety of software engineering tools to be developed and offered to the community for use. Based on the assessment of requirements techniques provided in Section 5, we believe that SREM has the best overall combination of capabilities to assure a positive impact in reducing the cost and risk of software development.

6.6 RECOMMENDATIONS

If the decision is made to implement SREM as a software tool within the Army, we recommend the following:

- Consider implementation of the RSL/REVS enhancements along the lines mentioned above.
- Develop a strategy for implementation of SREM for software requirements development and verification (or both) including the necessary means to enforce its application.
- Establish a plan for introduction of the methodology for both practitioners and managers, and for periodic SREM application training.
- Develop the capability to automatically produce software requirements in accordance with appropriate Army specification formats via RADX from a requirements data base built as the result of a SREM application, using "specification generators" capability, as described earlier.

In any of the efforts described above, the Army should coordinate its SREM-related efforts with BMDATC and with the U. S. Air Force at Rome Air Development Center (RADC). Both of these agencies are involved in efforts of related interest.

BMDATC continues to investigate advanced technology in the software area. This includes considerations of Distributed Data Processing (DDP), the development of System Software Requirements Engineering Methodology (SSREM) for producing the system level software requirement from the top level system specification and the development of a Software Design Engineering Methodology (SDEM) for software design. These last two will be SREM-like approaches with a methodology, a specific language, and a set of automated tools to support the effort. Both will be integrated with SREM so that the system level software requirements data base created by the SSREM will be directly useable for SREM application in developing the software requirements specification. Similarly, the data base developed during the application of SREM will feed directly into the SDEM effort. When completed, an integrated, consistent method for accomplishing all the engineering effort necessary to transit from the System Specification to the point of coding the software design is intended. Consequently, efforts in these areas should be of continuing interest during their development.

RADC has apparently decided to use SREM as a standard tool for the development of C³I systems within the U. S. Air Force. They have recently issued an RFP, the objective of which is "to develop the methodology, tools, and documentation required for a software requirements specification and analysis capability which can generate and validate a formal equivalent of the Computer Development Specification (MIL-STD 490 type B5) as used in the acquisition of computer systems embedded within Air Force C³I systems". This effort will provide for the study and development of enhancements to overcome some of the limitations described earlier in this section. It also will provide for improvements to the methodology, to REV5 software, for updating the existing SREM documentation, and for improving the educational materials to incorporate the enhancements produced under the contract. Clearly, a harmonization of AIRMICS efforts with those of the Air Force and with BMDATC makes good economic sense for all concerned.

7.0 REFERENCES

1. Alford, M. W. and Lawson, J. T., "Software Requirements Engineering Methodology Development", TRW Report 32697-6921-002, Huntsville, Alabama, 15 March 1979.
2. Alford, M. W., "Software Requirements Engineering Methodology (SREM) at the Age of Four", COMPSAC Proceedings, pp 866-874.
3. "A New Approach for Software Success", TRW Pamphlet, Huntsville, Alabama, undated.
4. Browne, P. H., Jr., Hitt, G. C., and Smith, R. W., "Utilization of SREM in IHAWK/TSQ-73 Requirements Development", TRW Report 27332-6921-034, Huntsville, Alabama, September 1978.
5. Baker, L. et al., "Specification Tools Environment Study", TRW Report 35983-6921-002, Huntsville, Alabama, 19 December 1980.
6. "Detailed Functional System Requirement (DFSR) - Volume IV, Standard Army Maintenance System (SAMS) - Retail Level, Maintenance Operations Management (MOM), (SAMS-1)", TM 38-L71-2, United States Army Logistic Center, Ft. Lee, Virginia, March 1979.
7. Dyer, M. E., et al., "REVS Users Manual, SREP Final Report - Volume II", TRW Report 27332-6921-06, (With Revisions A and B), Huntsville, Alabama, 1 August 1977.
8. Furbush, W. J., Alford, M. W., Boling, L. T., "Software Engineering Systems for Distributed Data Processing", TRW Preliminary Users Manual 36551-6921-008, Huntsville, Alabama, 11 February 1981.
9. Lawson, J. T., Gunther, L. J., Williams, R. L., "SREM Development for Navy Business Data Processing", TRW Report 32620-6921-002, Huntsville, Alabama, September 1979.
10. Sims, D. M., et al., "Advanced Data Processing Concepts", TRW Report 34673-6921-017, Huntsville, Alabama, 20 February 1981.

END

DATE
FILMED

7-8-1

DTIC